Delphi Prism e ASP.NET

Reaproveite seus conhecimentos Win32 / Desktop para criar aplicações Web – Parte 7

Clube Delphi

Compressão de arquivos

Conheça o componente TZipMaster

Mini-Curso

Padrões de Projeto – Parte 3

Ano 9 :: Edição 124 :: R$ 11,90

DevMedia

Controle de versão com Delphi XE

Brinde

Comprando esta edição ganhe 3 vídeo-aulas:

- Delphi Prism – DataViews e Banco de Dados
- Delphi Prism – DataSet e XML com SQL Server
- Delphi Prism – Stored Procedures: Múltiplos ResultSets

Quick Update

Frameworks inovadores

Delphi XE

Overview das novidades

ISSN 1517990-7

Dificuldades em encontrar conteúdo sobre .Net para iniciantes?

A DevMedia simplifica pra você!

easy .net magazine

Easy .Net Magazine:

A primeira revista digital feita especialmente para quem está dando os primeiros passos em .Net!

Você encontrará exemplos e modelos fáceis e práticos, noções introdutórias e conceitos básicos.

Conheça as vantagens:

- Vídeos e apresentações "embutidos" nas páginas;
- Ilustrações, diagramas e modelos apresentados em PowerPoint;
- Entrevistas em vídeo;
- Códigos-fonte que podem ser copiados com CTRL+C/CTRL+V.

A Easy.Net não é feita em PDF ou Flash!
Ela utiliza um aplicativo 100% online.

DevMedia group

Acesse agora a edição Nº 1 totalmente grátis e comprove

www.devmedia.com.br/easynet

Easy.Net Magazine, seu curso inicial de .Net!

RENOVE JÁ!

Sua assinatura pode estar acabando

Renovando a assinatura de sua revista favorita você ganha brindes e descontos exclusivos.

Faça a renovação de sua assinatura agora mesmo.

www.devmedia.com.br/renovacao ou (21)3382-5038

DevMedia

Hospede seus sites de forma simples, rápida e ilimitada.

Plano **Básico**

- 1 domínio principal
- 50 domínios hospedados
- Transferência **ilimitada**
- Espaço **ilimitado**
- 10 contas de e-mail
- PHP, ASP e ASP.NET
- Painel de controle Plesk
- Suporte 24hs

Tudo isso por apenas

R$ **15,90** por mês
pagamento trimestral

Assine agora mesmo e descubra por que Hospedix é a hospedagem de sites preferida entre os profissionais.

- Conheça também nossos planos profissionais com registro de domínio grátis e isenção de taxas anuais enquanto for cliente.

HOSPEDIX
HOSPEDAGEM PROFISSIONAL
www.hospedix.com.br

30 DIAS GRÁTIS

Ao assinar, digite o código de desconto exclusivo para leitores desta revista e ganhe 30 dias inteiramente grátis.

RVSTMEDIA

As promoções anunciadas não são cumulativas com outras promoções e são válidas apenas para clientes Hospedix ativos.

Sumário

Quick Update

05 – Frameworks inovadores
[*Paulo Quicoli*]

Delphi

Novidades

08 – Delphi XE
Overview das novidades
[*Daniel Sobrinho Laporte*]

Boa Ideia

12 – Compressão de arquivos
Conheça o componente TZipMaster
[*Vitor Luiz Rubio*]

Boas Práticas

Boas Práticas – Novidades

25 – Subversion
Controle de versão no Delphi XE
[*Rafael Stavarengo*]

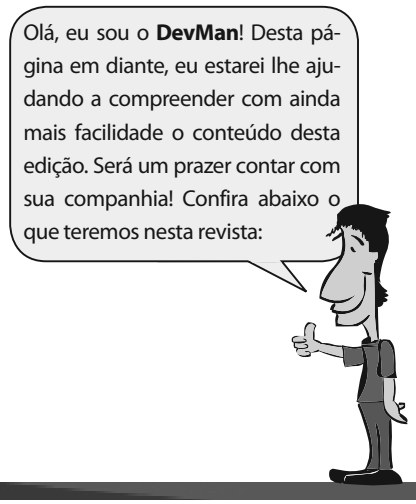
Boas Práticas

33 – Design Patterns - Parte 3
Usando Factory, Mediator e Singleton para construir frameworks abstratos de acesso a dados
[*Vitor Luiz Rubio*]

Easy Delphi

Easy Delphi - Mini-Curso – Tutorial – Web – Novidades

45 – Delphi Prism e ASP.NET
Reaproveite seus conhecimentos Win32 / Desktop para criar aplicações Web – Parte 7
[*Guinther Pauli*]



Brinde na web desta edição - 3 vídeos! Confira:

3 Vídeos

- **Curso Delphi Prism – DataViews e Banco de Dados**
  http://www.devmedia.com.br/post-17824-DataViews--Banco-de-Dados-Curso-Completo-Delphi-Prism-Desenvolvimento-Web-Parte-65.html
- **Curso Delphi Prism – DataSet e XML com SQL Server**
  http://www.devmedia.com.br/post-17680-DataSet-e-XML-com-Banco-de-dados-SQL-SERVER-Curso-Completo-Delphi-Prism-Desenvolvimento-Web-Parte-62.html
- **Curso Delphi Prism – Stored Procedures: Múltiplos ResultSets**
  http://www.devmedia.com.br/post-17601-StoredProcedures--Multiplos-resultsets-Curso-Completo-Delphi-Prism-Desenvolvimento-Web-Parte-59.html

Para acessar os vídeos utilize os seguintes dados
Login: **DVM.PL** *Senha:* **AJN47H3**

Gostou das vídeo aulas? O portal *www.devmedia.com.br* possui mais de **2 mil vídeo aulas** e **dezenas de cursos online** sobre desenvolvimento de software! Agora você pode comprar as vídeo aulas que preferir e fazer sua própria combinação de vídeos! Saiba mais em *www.devmedia.com.br/creditos*

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Para votar, você vai precisar do código e senha de banca desta edição, que é:
Login: **DVM.PL** Senha: **AJN47H3**

Clube Delphi

Ano 9 - 124ª Edição 2010 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Paulo Quicoli
pauloquicoli@devmedia.com.br

Daniel Laporte
daniel.sobrinho@devmedia.com.br

Rodrigo Carreiro Mourão
rodrigocarreiro@rmfactory.com.br

Equipe Editorial
Vitor Luiz Rubio
vitorrubio@gmail.com

Rafael Stavarengo
faelsta@gmail.com

Jornalista Responsável
Kaline Dolabella - JP24185

Capa e Diagramação
Romulo Araujo
romulo@devmedia.com.br

Coordenação Geral
Daniella Costa - daniella@devmedia.com.br

Revisão e Supervisão
Thiago Vincenzo - thiago.v.ciancio@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Isabelle Macedo e Uellen Goulart – Atendimento ao Leitor
www.devmedia.com.br/mancad
(21) 3382-5038

Kaline Dolabella – Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 3382-5038

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

Cristiany Queiróz
publicidade@devmedia.com.br

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

EDITORIAL

O Delphi XE foi lançado pela Embarcadero, trazendo com ele inúmeras novidades, que você passa a acompanhar a partir desta edição aqui na Revista ClubeDelphi. Pessoalmente considerei a nova versão algo como um novo "Delphi 7". Para quem lembra, esta versão do Delphi foi a mais utilizada de todos os tempos, porém se observar apenas manteve toda a sua base do Delphi 6 introduzindo ferramentas de terceiros já consolidadas no mercado. De fato, foram poucas novidades nativas. No Delphi 6, tivemos o BizSnap (basicamente suporte à Web Services e SOAP), inclusão da CLX (biblioteca cross platform), WebSnap (framework que turbinou o WebBroker e permitiu um desenvolvimento mais visual baseado em adapters e scripts). Foi no Delphi 6 que surgiu também a primeira versão do dbExpress para Win32, já que ele tinha aparecido no Kylix 1 um pouco antes. O Delphi 7 simplesmente adicionou ao já consolidado e glorioso Delphi 6 as ferramentas de terceiros IntraWeb, Rave Reports, Modelmaker e Bold, além claro de alguns incrementos menores na RTL e VCL, menos significativos. O Delphi 7 trouxe o primeiro preview do compilador para .NET. O Delphi XE é o novo Delphi 7. Ele mantém tudo o que tivemos de novo no Delphi 2009 / 2010 e o turbina nada mais nada menos com aquelas que considero as melhores ferramentas de terceiros para Delphi da atualidade. O AQTime é uma obra prima, você pode fazer um checkup no seu EXE rodando, como se fosse fonte (desde que seja compilado com informações de depuração). Monitore o tempo de chamada de procedimentos, detecte gargalos, alocação e liberação de memória (e memory leaks) e muito mais. O CodeSite é um "debugger" excelente, um "depurador dos pobres" que ficou rico. Quantas vezes utilizamos os famosos ShowMessage, WriteLn ou Response.Write (na Web) para indicar em tela que determinado ponto do programa foi executado, ou mesmo para inspecionar o valor de uma variável em memória em certo ponto da execução? Agora temos uma ferramenta extremamente profissional para este propósito. O FinalBuilder é a melhor aplicação para automação de Build que existe no mercado, em minha opinião. Em uma situação real, foi o FinalBuilder que reduziu drasticamente o trabalho que tínhamos em um projeto onde existiam mais de 5 mil units e forms. Diariamente o aplicativo compilava centenas de componentes clientes, bibliotecas servidoras DataSnap, formulários, gerava pacotes BPL, os executáveis, enviava tudo para seus devidos lugares (fontes no repositório no StarTeam e aplicações para ambiente de homologação), gerava DOCs atualizados do framework e enviava LOGs de tudo para a equipe. Em uma equipe de 100 pessoas, este era literalmente nosso 101º membro do time. O Subversion é a ferramenta mais popular para o controle de versão de código fonte, basta mencionar que agora no Delphi XE a Embarcadero integrou a ferramenta ao IDE. Temos também o início do suporte à plataforma de Cloud Computing da Microsoft, Windows Azure.

Já nesta edição você conta com uma matéria do Daniel Laporte dando um overview sobre as novidades. O Rafael Stavarengo inicia a cobertura das melhorias e novas ferramentas, destacando o Subversion. E mais, o Vitor Rubio traz duas matérias, uma sobre compressão de arquivos (exibindo inclusive dois excelentes exemplos práticos de uso do componente Zipmaster) e a finalização da série sobre Design Patterns. Finalmente, a última parte do minicurso onde mostro como usar o Delphi Prism para construir aplicações Web de alta performance com ASP.NET, reaproveitando conhecimentos em Delphi Win32 / Desktop.

Boa leitura e sucesso com o Delphi XE.

Guinther Pauli

guinther@devmedia.com.br

Certificado Delphi 3,5,6,7,2005,2006,Web,Kylix

Microsoft Certified: MCP,MCAD,MCSD.NET,MCTS,MCPD (ASP.NET,C#)

guintherpauli.blogspot.com - twitter.com/guintherpauli

ClubeDelphi PLUS

DevMedia

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS.
Para mais informações sobre o pacote PLUS, acesse:
http://www.devmedia.com.br/clubedelphi/portal.asp

Quick Update: Frameworks inovadores

Paulo Quicoli
pauloquicoli@gmail.com
Editor técnico das revistas ClubeDelphi, .NET Magazine e WebMobile Magazine. Formado em processamento de dados pela FATEC-TQ. Atua como analista e desenvolvedor da ControlM Informática (www.controlm.com.br) e professor de linguagem de programação do curso técnico em processamento de dados do SENAC/Jaboticabal. Utiliza Delphi desde sua primeira versão, entusiasta do desenvolvimento orientado a objetos, tem publicado vários artigos sobre o assunto. Blog: www.pauloquicoli.spaces.live.com

Sem dúvida grandes plataformas como .NET e Java apresentam diversos frameworks que auxiliam no desenvolvimento puramente orientado a objetos. O Delphi por ter uma característica altamente RAD, apesar de ser totalmente orientado a objetos, não incentiva seus usuários a essa prática. Contudo existem desenvolvedores Delphi ao redor do mundo que querem utilizar a orientação a objetos de forma tão prática como os desenvolvedores .NET ou Java fazem e para isso não medem esforços no desenvolvimento para "converter" frameworks existentes para a linguagem Delphi. Vamos conhecer hoje dois projetos muito interessantes, um para persistência de objetos e outro um container e injeção de dependência.

Projeto DORM

Esse é o acrônimo para Delphi Object Relational Mapper. Esse framework tem o papel de fazer uma ponte entre seus objetos e o banco de dados. Segue a implementação do padrão Data Mapper porém, seguindo características do mais famoso ORM do mercado, o NHibernate. O criador desse projeto é o italiano Danielle Teti, que pude encontrar no último Delphi Conference e conversar sobre o projeto.

Segundo ele, a ideia de se basear no NHibernate não é fazer uma cópia exata, mas sim, pegar o que há de melhor em sua implementação e aplicar em uma versão Delphi. Com isso em mente, para configurar o DORM e já mapear, por exemplo, duas classes TPerson e TPhone é necessário criar um arquivo chamado dorm.conf, no formato JSON, veja **Listagem 1**.

Nas primeiras sete linhas estamos configurando o acesso ao banco de dados, das linhas 8 a 11 configuramos o comportamento do framework. Estamos informando onde estão as classes a serem mapeadas e definimos também uma classe que fará o Log das operações. Na sequência (linhas 12 a 47) fazemos o mapeamento das classes TPerson e TPhone. Se compararmos esse mapeamento com o mapeamento XML do NHibernate veremos grandes similaridades conceituais. Uma vez configurado e mapeado, podemos fazer um uso semelhante ao da **Listagem 2**.

As principais características do framework são:

- Arquivos externos de mapeamento (formato JSON);
- Persistência de PODOs (Plain Old Delphi Objects), qualquer TObject pode ser persistido;
- Suporte para relacionamentos 1:1 e 1:M, ainda sem M:M;
- Suporte para Lazy Loading;
- Suporte para IdentityMap;

Listagem 1. Arquivo de configuração

```
1.  {
2.    "persistence": {
3.    "database_adapter":"dorm.adapter.Firebird.TFirebirdPersistStrategy",
4.    "database_connection_string":"127.0.0.1:C:\\MyProjects\\
      DORM\\experiments\\dorm.fdb",
5.    "username":"sysdba",
6.    "password":"masterkey"
7.    },
8.    "config": {
9.       "package":"dorm.bo.Person",
10.      "logger_class_name":"dorm.loggers.FileLog.TdormFileLog"
11.  },
12.  "mapping":
13.      {
14.      "TPerson":
15.        {
16.        "table":"people",
17.        "id": {"name":"guid","field":"guid","field_type":"string",
            "size": 100,"default_value":""},
18.        "fields":[
19.        {"name":"firstname","field":"first_name",
           "field_type":"string","size": 100},
20.        {"name":"lastname","field":"last_name","field_type":"string",
           "size": 100},
21.        {"name":"age","field":"age","field_type":"integer"},
22.        {"name":"borndate","field":"born_date","field_type":"date"}
23.         ],
24.      "has_many":[{
25.        "name":"Phones",
26.        "class_name":"TPhone",
27.        "child_field_name":"guid_person",
28.        "lazy_load": false
29.      }],
30.      "has_one": {
31.        "name":"car",
32.        "class_name":"TCar",
33.        "child_field_name":"guid_person"
34.      }
35.    },
36.    "TPhone":
37.    {
38.      "table":"phones",
39.      "id": {"name":"guid","field":"guid","field_type":"string",
          "size": 100,"default_value":""},
40.      "fields":[
41.        {"name":"number","field":"number","field_type":"string",
           "size": 100},
42.        {"name":"kind","field":"kind","field_type":"string",
            "size": 100},
43.        {"name":"guid_person","field":"guid_person",
            "field_type":"string","size": 100}
44.        ]
45.    }
46.  }
47. }
```

Listagem 2. Usando o DORM

```
procedure SalvarPerson;
var
 p: TPerson;
 t: TPhone;
begin
  Session := TSession.Create;
  Session.Configure(TStreamReader.Create('dorm.conf'));
  //método estático, retorna um objeto populado
  p := TPerson.NewPersona;
  try
    t := TPhone.Create;
    t.Numero := '555-7765123';
    t.Kind := 'Casa';
    p.Phones.Add(t);
    //salva pessoa e telefones
    Session.Save(p);
  finally
    Session.Commit;
  end;
end;
```

Nota do DevMan

O JSON (JavaScript Object Notation) é uma espécie de padronização para transporte de informações entre ambientes heterogêneos e que tem como linguagem o JavaScript. Padrão, transporte, ambiente heterogêneo, tudo isso nos remete à XML, então vem a questão: Por que JSON e não XML? O principal motivo seria pelo fato da notação JSON ser extremamente mais simples do que a XML. Não existem tags diversas nem regras complexas para escrita e muito menos mecanismos complexos de parser.

Nota do DevMan

POCO em .net quer dizer Plain Old CLR Object, significa que nossas entidades de negócio devem ser o mais simples possível, sem nenhuma dependência de ferramentas externas. Equivalem ao POJO do Java. No Delphi, de forma análoga, temos as classes PODO (Plain Old Delphi Objects). São objetos que simplesmente herdam diretamente de TObject e não necessitam herdar de outra classe para serem persistidos. Mas qual é o benefício disso? Por que se fala tanto em ter uma camada de domínio da aplicação, desacoplada do resto? A resposta é simples. O domínio da aplicação, onde temos o modelo de entidades, assim como a modelagem das regras de negócio, é como o motor de um carro. E nós queremos construir um carro que, quando quebrar uma roda, seja necessário trocar apenas a roda e não o motor. Se você deixar o motor do carro acoplado à roda, toda vez que tiver que trocar a roda terá que trocar o motor também. É isso que acontece em muitas aplicações hoje em dia. Muitos projetos precisam ser refeitos inteiramente porque o domínio da aplicação está espalhado entre as camadas e depende de todo tipo de tecnologia, como banco de dados, ferramentas de persistência e até da camada de apresentação. E como as tecnologias não param de surgir, é importante que comecemos a planejar nossas aplicações para sobreviver a essas mudanças, senão vamos viver recriando o motor dos nossos projetos, toda vez que surge uma roda melhor no mercado. POCO / PODO é uma boa prática voltada para isso.

• Suporte para customizar buscas. Você pode usar SQL se for o caso;
• Suporte completo a operações CRUD;
• Transações;
• Sistema de log incluso para registrar todo SQL ou ação realizada pelo framework;
• Suporte a vários tipos de banco de dados;
• Cache para RTTI.

Mas como nem tudo são flores, ainda não existe um release oficial, nem mesmo o código fonte está liberado. Em conversa com o próprio Danielle, ele e seu sócio Fabrichio ainda precisam implementar alguns recursos que eles desejam e ainda

vão decidir como o framework será distribuído, se free ou se virá integrado com a próxima versão do Delphi.

Projeto Delphi DIContainer

A injeção de dependência é um padrão de desenvolvimento utilizado na orientação a objetos quando queremos manter baixo o nível de acoplamento entre diferentes partes de uma aplicação. Em Delphi não tínhamos até então nenhum container que implementasse isso para nós. Se desejássemos fazer uso do padrão, era necessário fazer o controle manual. O mesmo Danielle do projeto DORM também solucionou esse problema implementando o Delphi DIContainer. Esse framework tem o objetivo de centralizar a localização de serviços (interfaces) e suas implementações. Vejam o trecho de código da **Listagem 3**.

No exemplo da **Listagem 3** estamos trabalhando apenas com classes, mas o mesmo poderia ter sido feito com interfaces. Nós instanciamos o container, registramos nele um tipo que será instanciado sob o padrão Singleton, ou seja, o container irá garantir que só exista uma única instância do tipo TService1. Na sequência passamos para a variável s1 a instância retornada pelo container. Então atribuímos um valor para sua propriedade Message. Continuando pegamos novamente através do container um instância de TService1, porém como ela foi registrada sob o padrão Singleton, a mesma instância criada anteriormente será retornada à variável s2. Ao imprimir a propriedade Message das variáveis veremos que ambas têm o mesmo valor.

Além de implementar o controle de instâncias, este container também controla a injeção automática de dependência via construtores (veja o QuickUpdate da edição 123).

Como podemos ver esse é um ótimo avanço para o desenvolvermos software orientado a objetos de qualidade dentro do Delphi. O Delphi DIContainer já possui código fonte liberado.

Listagem 3. Uso de Delphi DIContainer

```
var
  DIContainer: TDIContainer;
  s1, s2: TService1;
begin
  DIContainer := TDIContainer.Create;
  DIContainer.AddComponent(TService1, TDIContainerInit
   Type.Singleton);
  s1 := DIContainer.GetComponent('ServiceTestObjectsU.TService1')
      as TService1;
  s1.Message := 'I"m the first message';
  s2 := DIContainer.GetComponent('ServiceTestObjectsU.TService1')
      as TService1;
  WriteLn(s1.Message);
  WriteLn(s2.Message);
end;
```

Blog do Danielle Teti
http://www.danieleteti.it

Download do Delphi DIContainer
http://code.google.com/p/delphidicontainer/

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Faça um *upgrade* em sua carreira

Em um mercado cada vez mais focado em qualidade, ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre ouutros, pode ser a diferença entre conquistar ou não uma boa posição profissional. Sabendo disso a DevMedia lançou uma publicação totalmente especializada em Engenharia de Software. Todos os meses você pode encontrar artigos sobre Metodologias Ágeis; Metodologias tradicionais (document driven); ALM (application lifecycle); SOA (aplicações orientadas a serviços); Análise de sistemas; Modelagem; Métricas; Orientação à Objetos; UML; testes e muito mais. **Assine Já!**

DevMedia

Delphi XE

Overview das novidades

O Delphi 2009 (e mais recentemente o Delphi 2010) trouxe diversas melhorias consideráveis e até mesmo revolucionárias. Pode-se citar aí o suporte a Generics, Unicode, Natural Input, Multitouch e Gestures, novo DataSnap, driver de conexão nativa com o Firebird através do DBExpress, suporte a JSON, REST, novas units, disponibilidade do MIDAS para customização, melhoria na RTTI entre muitas outras novidades.

A Embarcadero lança agora o Delphi XE. O conjunto de soluções XE envolve o Delphi Prism XE, RAD Studio XE e RADPHP XE, soluções para .NET, Win32 e PHP. Com todas essas novidades a ClubeDelphi exibirá em suas próximas edições, artigos abordando todas as novidades das mais novas versões das soluções XE da Embarcadero. Neste artigo faremos um pequeno overview de tudo que nos espera nessas próximas edições e principalmente, o que o novo produto pode trazer de benefício para nós desenvolvedores.

Além de todas as novidades que tivemos a partir de quando a Embarcadero assumiu o Delphi, essa versão vem com um diferencial em relação a todas as outras versões anteriores. Nesta versão foram selecionados alguns dos aplicativos já utilizados e muito conhecidos por grande parte da comunidade, sendo estas, consideradas muitas vezes como as melhores ferramentas de terceiros para Delphi em seus respectivos segmentos. Com isso temos grandes aliados no desenvolvimento. A grande vantagem é que todas essas ferramentas agora são integradas ao IDE, não precisam ser usadas como utilitários externos (apesar de possível).

Cloud Computing

Como já é do conhecimento de todos, a nova onda tecnológica e provavelmente tendência futura, aborda o tema de computação das nuvens. Com a introdução desta nova metodologia de desenvolvimento, teremos a possibilidade de disponibilizar nossas aplicações todas independentes, rodando na Web, ou armazenando dados na cloud. Como a tecnologia é nova e vem crescendo exponencialmente, um dos grandes desafios que já se encontram traçados nos planos de empresas inovadoras, diz respeito à Cloud Computing, onde o Google propõe desenvolver o primeiro Sistema Operacional totalmente em "nuvem".

Como já era de se esperar, a Embarcadero tratou de se adequar a esta nova tendência. Uma das principais novidades revolucionárias da nova versão XE é a possibilidade de criar servidores DataSnap (agora também disponíveis em C++ a partir da versão XE) integrando-se à tecnologia Windows Azure da Microsoft. Toda esta comunicação é realizada através de REST e TCP/IP. Para suporte a Cloud Computing, a partir desta versão foi disponibilizada a paleta

Resumo DevMan

De que trata o artigo:

Abordar um pequeno overview sobre a mais nova versão do Delphi, chamada de Delphi XE, ilustrando um pouco de suas novidades apenas para introduzir ao leitor a essa nova evolução e porque não dizer revolução, que o Delphi vem tendo.

Para que serve:

Demonstrar a evolução da mais conhecida ferramenta da Embarcadero ilustrando evoluções no IDE, ferramentas de auxílio ao aumento de produtividade, controle e organização de processos, Profiling de código, controle de versionamento entre muitas outras coisas.

Em que situação o tema é útil:

Uma boa pedida para se inteirar das novidades da nova versão. Para quem está pensando em atualizar sua antiga versão do Delphi 7, ou mesmo quem já possui uma mais recente, este artigo irá cobrir todas as novidades que influenciarão na aquisição do novo IDE.

Delphi XE:

Lançado recentemente, o Delphi XE é a evolução do Delphi 2010, que foi considerada a melhor versão desde o Delphi 7, ainda na época da Borland. Serão abordadas neste artigo as evoluções desenvolvidas pela Embarcadero, isso inclui suporte à tecnologia Cloud Computing através da plataforma Windows Azure, integração do Subversion (SVN) com o IDE e todas as ferramentas que foram acrescentadas ao produto como CodeSite, AQTime, FinalBuilder entre muitas outras novidades.

Windows Azure, especialmente para a tarefa de nuvem (**Figura 1**).

Uma grande evolução ainda se tratando do novo DataSnap é a chamada "ProxyClient", que trata-se de uma camada a mais que faz com que seja disponibilizado o consumo de métodos de um servidor DataSnap em diferentes linguagens como: JavaScript, Object Pascal, PHP, C++ Builder entre outras. Isso faz com que seja ainda maior a capacidade de comunicação e interoperabilidade entre ambientes heterogêneos, permitindo esta interatividade entre tecnologias através de diferentes "Clients", ou seja, um Client em PHP pode se comunicar com um servidor DataSnap construído em Delphi o C++ e hospedado em nuvem através da tecnologia Windows Azure.

Os componentes do Delphi XE permitem trabalhar com dados armazenados nas nuvens no Windows Azure, incluindo Table, Blob e Queue (veja **Tabela 1**). Além disso, possui facilidades para deployment para os serviços de cloud computing da Amazon (Amazon EC2).

Controle de versão - SubVersion

Uma das questões mais interessantes e importantes, utilizadas por grande parte dos desenvolvedores, não só que utilizam Delphi, mas de um modo geral, é o controle de versões. O controle de versões até então sempre foi conhecido e utilizado pelos desenvolvedores, tendo uma grande gama divida entre o Jedi VCS e Subversion. Muitos desenvolvedores também utilizam este recurso na ferramenta StarTeam, que também permite este controle.

Para quem nunca utilizou e apenas ouviu falar neste assunto, o controle de versionamento é um grande aliado quando trabalhamos em equipe em um mesmo código-fonte. Com este recurso, é possível gerenciar todas as alterações de projetos realizadas pelos membros da equipe, sendo possível visualizar logs de alterações e realizar tarefas como merge de units que sofreram alteração, criação de repositórios para controle de versão e muito mais, fazendo assim com que o seu código-fonte mantenha-se sempre atualizado.

A partir da versão XE a ferramenta Subversion está inteiramente integrada ao IDE, como mostra um menu de contexto no Project Manager (**Figura 2**), permitindo realizar todas as operações de manipulação dos arquivos de forma muito mais rápida.

Componente	Funcionalidade
TAzureConnectionString	Usado para prover informações da conexão Azure como AccountKey, Account Name e protocolo. Cada um dos componentes da paleta Azure necessita de um TAzureConnectionString colocado no mesmo formulário para fornecer os parâmetros de conexão.
TAzureQueueManagement	Com o Windows Azure você pode ter muitas queues (mensagens) e cada queue pode ter muitas mensagens através da API REST. O componente TAzureQueueManagement é limitado a mostrar apenas as 100 primeiras mensagens de cada queue.
TAzureTableManagement	Trata-se de um componente um pouco mais complexo que o TAzureQueueManagement, mas de configuração idêntica. A diferença é que é próprio para a manipulação de dados (tabelas), como o próprio nome sugere.
TAzureBlobManagement	É utilizado para manipulação de Blobs (Binary Large Objects). Permite operações como criar container, copiar blobs, bloquear propriedades, criar blocos, criar "snapshots" (imagens) e paginar.

Tabela 1. Descrição dos componentes Windows Azure

Automação de Build - FinalBuilder

Dependendo do tamanho do sistema, uma compilação geral pode dar trabalho e tomar muito tempo. Processos comuns incluem: compilar biblioteca de componentes, aplicações clientes, aplicações

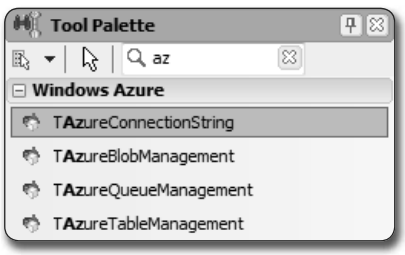


Figura 1. Paleta Windows Azure

Nota do DevMan

Windows Azure e Cloud Computing – Uma visão geral

Houve um tempo em que cada um tinha o seu próprio poço, de onde vinha a água para o consumo diário da casa, assim como também houve o tempo em que as pessoas precisavam de grandes geradores de energia movidos a diesel para ter energia elétrica em suas fazendas. Hoje, tanto a água quando a energia elétrica vêm, através de canos e fios, abastecer nossas casas e dar conta das necessidades diárias que temos desses recursos. Você por acaso sabe o caminho que a energia elétrica percorre até chegar à sua casa? Todo o processo que envolve essa geração e abastecimento, desde o princípio lá na usina (provavelmente hidroelétrica) até o ponto de energia onde você liga o seu note? E a água? Não, a maioria de nós não tem nenhuma pista de como tudo isso funciona, e também não damos a mínima para isso. O fato é que consumimos água e energia elétrica, além de muitos outros recursos, sem nos preocupar de onde vem e como foi chegar ali. O princípio por trás da computação em nuvem, e por consequência o Windows Azure, suportado agora no XE, é o mesmo. O que a Microsoft está oferecendo com o Windows Azure não é apenas um provedor gigante para hospedar suas aplicações, ela está oferecendo recursos de processamento e armazenamento, como se fosse energia elétrica ou água potável. São enormes data-centers distribuídos pelo mundo, no que podemos literalmente chamar de fazendas de servidores, formando uma rede enorme de hardware. Quando usamos uma aplicação que está publicada no Windows Azure, estamos consumindo processamento e memória dessa enorme estrutura. A forma como todo esse processamento e armazenamento é cobrada de você também segue um princípio muito parecido com o dos serviços de energia elétrica e água, basicamente de acordo com o seu consumo.

servidoras, ajustar arquivos INI, ajustar registro do Windows, rodar testes, gerar instalador, disponibilizar na Web, enviar os fontes atualizados a um repositório de controle de versão (isso sem contar com as chances de erro). Essas são tarefas rotineiras na vida de um desenvolvedor. Já imaginou se tudo isso pudesse ser automatizado?

O FinalBuilder é uma ferramenta que realiza uma automação completa no processo de Build de aplicativos, possibilitando a criação de "constraints" de validação, agendamento de compilação, interação com seu código-fonte, criação de scripts, formatação de código etc.

Vale lembrar que a ferramenta está embutida na própria instalação do RAD Studio XE, mas para você que ainda utiliza versões anteriores ao XE, pode realizar o download da ferramenta no endereço que se encontra na sessão Links. Um detalhe muito interessante é que o FinalBuilder não faz este processo apenas para projetos em Object Pascal com o compilador Delphi, mas sim com diversos outros compiladores de diversas outras linguagens. A **Figura 3** ilustra a tela de entrada da ferramenta.

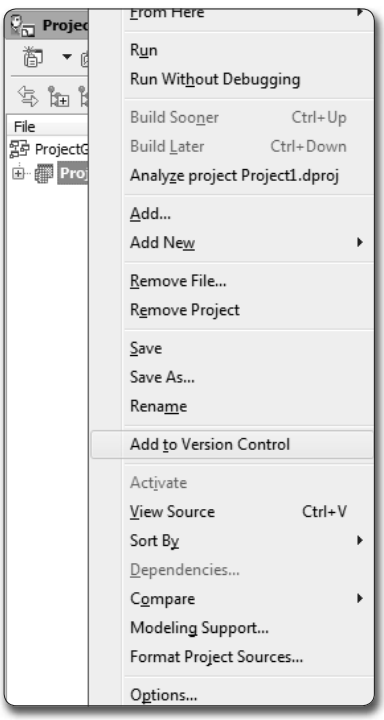


Figura 2. Opção do Subversion integrado ao IDE

Profiling – AQTime

O Delphi XE vem com uma poderosa ferramenta de Profiling de código chamada AQTime. Para quem ainda não conhece, a técnica de Profiling consiste em inspecionar/investigar seu código-fonte e detectar possíveis pontos falhos, problemas entre outras coisas, sendo muito utilizada principalmente na área de engenharia de software. É justamente esta a proposta da ferramenta AQTime, que é totalmente integrada ao IDE do Delphi através do menu de mesmo nome. Veja na **Figura 4** um exemplo de Profiling de uma pequena aplicação.

A **Figura 5** ilustra os diferentes tipos de Profiling que o AQTime permite que sejam realizados no seu software. É importante citar que estes Profilings estão disponíveis de acordo com o tipo de licença do AQTime. Os Profilers existentes permitem detectar gargalos de execução na aplicação (por exemplo qual linha de código está levando mais tempo para ser executada, ou ainda detectar memory leaks, inspecionar instruções SQL, liberação de objetos, carregamento de bibliotecas, alocação de memória etc.

CodeSite

Uma outra novidade do novo RAD Studio XE é o CodeSite (**Figura 6**). O CodeSite é uma ferramenta que é utilizada para realizar logs avançados de código, o que pode adicionar muita facilidade na hora do Debug do seu projeto.

Imagine que você está desenvolvendo um aplicativo e as coisas não estão correndo como deveriam: o programa não faz o que deveria fazer, você tem muita dificuldade para seguir o código usando o debugger integrado do Delphi, pois os métodos se entrelaçam e eventos são ativados durante a execução, ou mesmo o erro está no meio de uma rotina de reescrita da tela (nesse caso, quando o debugger é ativado a tela é coberta e a rotina é chamada novamente), ou mesmo sua aplicação é multithreaded, onde diversas tarefas são executadas simultaneamente (por exemplo aplicações Web ou servidores de aplicação DataSnap).

Normalmente, a primeira opção quando isso acontece é colocar uma chamada para "ShowMessage" para que a caixa de diálogo mostre o valor da variável. Quando a rotina com erro é chamada diversas vezes, isso é impraticável, pois

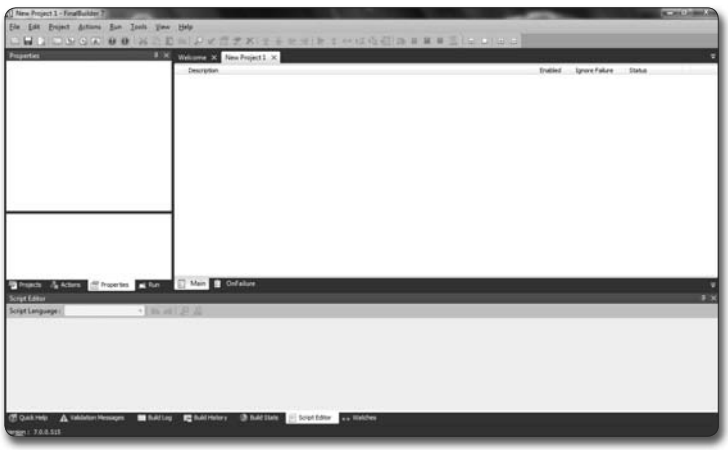


Figura 3. Janela principal do FinalBuilder

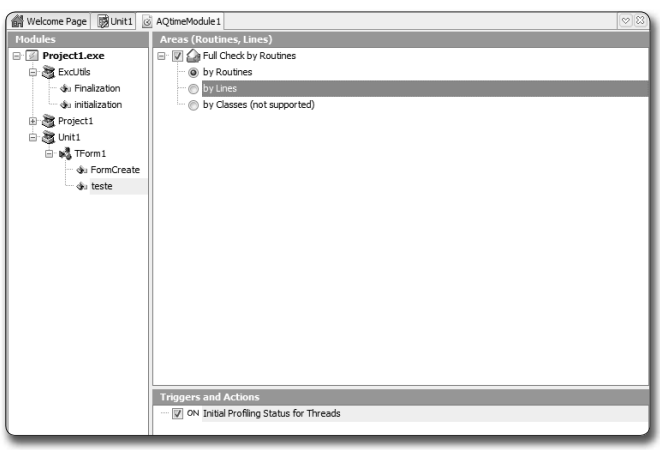


Figura 4. Profiling de código direto no IDE

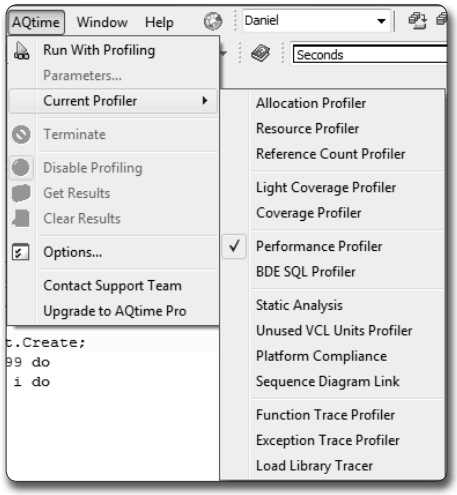


Figura 5. Diferentes tipos de Profiling

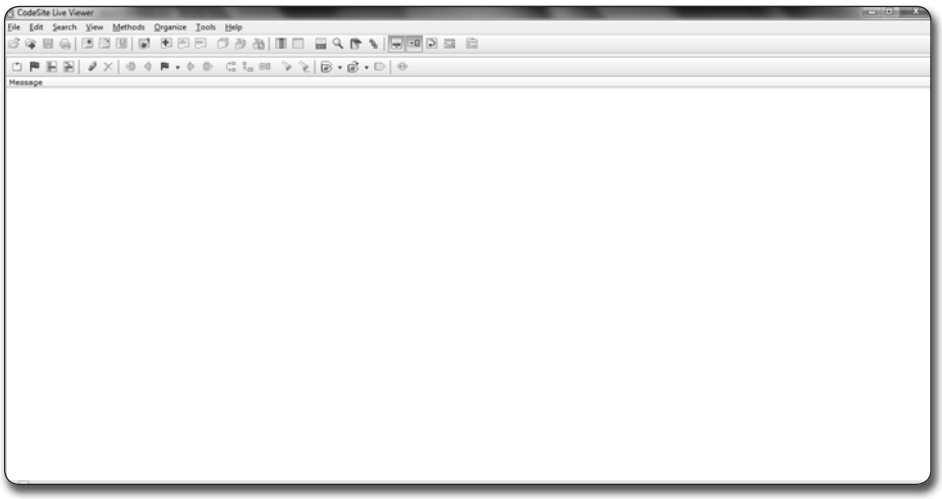


Figura 6. CodeSite

temos que clicar no botão OK cada vez, além de interromper a execução, escondendo bugs e introduzindo outros.

Passamos então para a segunda opção: criar um arquivo de log, usando WriteLn. Essa opção é mais interessante, pois não interrompe a aplicação e permite mostrar todas as informações desejadas. Esse enfoque funciona até o momento em que as funções começam a se entrelaçar ou quando temos que enviar informações mais complexas, como uma imagem. Uma solução para esses problemas pode ser o CodeSite.

O CodeSite é como um super gerador de arquivos de log, pois permite enviar dados de diversos tipos para o log, inclusive tipos personalizados. O log pode então ser visualizado localmente, no Viewer, gravado num arquivo, para ser visto em outro computador ou mesmo enviado para outra máquina. As mensagens podem ser de diversos tipos, sendo mostradas com ícones diferentes, podendo ser filtradas, para esconder mensagens desnecessárias.

Mais novidades e melhorias

Além de todas estas ferramentas, temos melhorias e novidades na VCL, RTL e STL, inserção de expressões regulares para RTL, melhorias nos tipos TStrings, Date/Time, foi adicionado o suporte à API OpenTools nas abas internas do IDE e também uma melhoria significante da performance do compilador.

Um outro ponto marcante no novo Delphi foi a preocupação da equipe com a qualidade do produto. A Embarcadero reporta que foram solucionados cerca de 1000 bugs só para esta versão do produto, mostrando que a empresa tem levado a qualidade e estabilidade da ferramenta como um ponto crucial e importante.

Conclusão

Este artigo mostrou um rápido overview das muitas novidades do novo IDE da Embarcadero. O leitor já pode ter uma breve ideia de tudo aquilo que o aguarda. Com estas ferramentas adicionadas ao nosso IDE favorito, podemos ter muito mais recursos para desenvolver aplicações de maior qualidade da melhor forma possível, tudo isso de forma muito simples e objetiva. A partir desta edição da Revista ClubeDelphi, você poderá acompanhar artigos detalhados sobre todas as novidades que apresentamos aqui.

Daniel Sobrinho Laporte
daniel.sobrinho@devmedia.com.br
daniel.laporte@gmail.com
www.twitter.com/daniel_laporte
Colunista e editor técnico da Revista ClubeDelphi. Graduado em Sistemas de informação e pós-graduado em Administração de Bancos de Dados pela Universidade de Franca. É analista de sistemas e desenvolvedor Delphi desde 2005 desenvolvendo aplicações cliente/servidor e multicamadas.

FinalBuilder
http://www.finalbuilder.com/home.aspx

AQTime
http://www.automatedqa.com/products/aqtime/

CodeSite
http://www.raize.com/DevTools/codesite/

Subversion
http://subversion.apache.org/

Lista de bugs corrigidos no Delphi XE
http://edn.embarcadero.com/article/40817

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Compressão de arq

Conheça o componente TZipMaster

Nas situações em que tudo já foi feito para aumentar a velocidade de uma transmissão de arquivos ou diminuir o espaço em disco que um arquivo consome, a solução pode ser a compressão de arquivos.

Existem vários tipos de compressão para vários tipos e formatos de arquivo. Por exemplo, mp3 é um tipo de compressão de áudio. Um arquivo mp3 é um tipo de arquivo de áudio wav que além de comprimido, são retiradas algumas frequências inaudíveis ao ouvido humano, para que se diminua o tamanho. Analogamente, JPG é um formato de imagem comprimido. Uma foto tirada com uma câmera de 7mpx, se fosse armazenada em bitmap, teria aproximadamente 20mb, enquanto que no formato JPG ela pode ocupar entre 5 e 7mb. Da mesma forma DivX, XviD entre outros são formatos de compressão de vídeo. Embora existam formatos e algoritmos específicos para compressão de certos tipos de dados, existem também algoritmos de compressão de uso geral.

A compressão pode se dar com perdas, onde partes dos dados são perdidos e o arquivo comprimido não pode mais voltar ao estado original. É o que acontece com MP3, DivX e JPG. Mas existem também os algoritmos de compressão sem perda, como o Zip, Rar e Arj.

Neste artigo aprenderemos uma maneira prática de criar arquivos .zip para envio ou transmissão imediata sem exigir que o cliente tenha ou saiba usar um compactador de arquivos. Para isso usaremos o componente TZipMaster. Seu download pode ser feito no site oficial, que pode ser encontrado na seção de links.

Streams

Este artigo aborda o uso de Streams. TStream é a classe base para tipos de objetos que podem ler ou salvar em vários tipos de armazenamento como arquivos, memória e outros mais. Mas o que é um TStream. Uma tradução bem aceita é que um TStream é algo como um fluxo de dados, informações. Sendo assim, esse fluxo possui um início e fim, e ao manipular essas informações sempre estaremos em algum desses pontos. Um TStream pode conter qualquer tipo de informação.

Resumo DevMan

De que trata o artigo:

Este artigo abordada a compressão de arquivos, quando ela é necessária e como fazê-la. Isso será feito usando o componente TZipMaster. Também aborda streams comprimidas e instaladores com arquivos comprimidos auto extraíveis.

Para que serve:

Por vezes é necessário armazenar um arquivo ou backup de modo a ocupar o menor espaço possível em disco. Outras vezes precisamos transferir um arquivo grande por e-mail, FTP ou até TCP/IP e precisamos de alguma forma diminuir o tamanho desse arquivo para que a transferência fique mais rápida. Outra serventia seria criar pacotes de patches já pré-configurados e com os arquivos corretos para que o usuário simplesmente instale.

Em que situação o tema é útil:

Sempre que for necessário comprimir, armazenar ou transferir um arquivo ele pode ser comprimido. TZipMaster nos permite criar programas de compressão de arquivos profissionais, como os famosos WinZip e PkZip. Pode ser útil para colocar um compressor embutido na sua aplicação e permitir que seu cliente envie logs do sistema sem custo adicional para ele.

Compressão de arquivos:

Embora o Delphi tenha uma biblioteca nativa para compressão de arquivos e streams, a Zlib, o uso do TZipMaster é muito mais fácil. Além disso, com o TZipMaster você pode fazer muito facilmente a compressão de pastas inteiras, o que necessitaria de algum trabalho extra usando Zlib. Além disso, você pode fazer programas de backup profissionais. Os arquivos comprimidos com TZipMaster podem ser facilmente abertos com o WinZip, PkZip ou 7zip. Neste artigo será mostrado um pouco sobre compressão de arquivos, compressão de streams e apresentado um programa para backup de bancos de dados Firebird.

Nota

Há diversas bibliotecas ou algoritmos de compressão. Exemplos comuns são ACE, RAR, ZIP, 7Z, TAR, ARJ. A maioria desses formatos possui uma API / SDK que também pode ser usada pelo Delphi. O Componente ZipMaster vem junto com uma DLL chamada DelZip190.dll que deve estar em um diretório facilmente encontrado pelo seu programa, seja no mesmo diretório do programa, seja na pasta system32 ou em outro diretório especificado pelo componente.

Nota

Nota: Se ao término da instalação não for possível encontrar uma aba chamada Delphi Zip na Tool Palette o instalador descomprimiu o source, compilou, mas não fez a instalação do componente. Dessa forma é necessário realizar a instalação manual. Isso pode ser feito abrindo a pasta de instalação e carregando o pacote ZMstr190D13.dpk, que é a versão para o Delphi XE e instalá-lo. É necessário também incluir no Libary Path e Search Path o caminho das units do componente.

uivos

Vitor Luiz Rubio

Além de ser uma classe base é também abstrata, por isso existem versões especializadas para cada tipo particular de armazenamento. Cada descendente de TStream implementa métodos que permitem transferir dados de forma adequada à sua finalidade. Por exemplo, se você utilizar um descendente de TStream destinado à utilização de arquivos, seus métodos são codificados para tratar somente esse tipo e de forma eficiente. O TStream também disponibiliza para as aplicações meios de localizar uma posição específica no conjunto de dados que ele contém e ainda fornece informações sobre esses dados, como seu tamanho e posição corrente durante a leitura.

A classe TStream é muito versátil, permite seu trabalho em conjunto com componentes para facilitar o uso nos mais diversos tipos de formulários. Como já mencionado, a classe TStream é abstrata e não deve ser instanciada, seus métodos são puramente virtuais que são sobrecarregados em classes descendentes. Aqui temos algumas dessas classes:

TFileStream – Deve ser utilizado para acessar informações contidas em arquivos. Um objeto do tipo TFileStream abre um arquivo e disponibiliza métodos para leitura e gravação;

TStringStream – Permite que informações sejam armazenadas no formato string, porém com recursos de I/O. Seu uso comum é de atuar como um objeto intermediário entre algum processo, contendo texto que será utilizado posteriormente;

TMemoryStream – Armazena informações em um buffer dinâmico de memória. Por estar em memória seu processamento é veloz, além de que também oferece mecanismos para manipulação das informações contidas. Assim como o TStringStream, é muito utilizado também como uma forma de objeto temporário, entre um processo e outro;

TBlobStream – Este atua diretamente sobre os campos BLOB de um DataSet do BDE. Também trabalha com os campos TMemoField e TGraphicField para leitura e escrita de dados;

TWinSocketStream – Disponibiliza meios para que aplicações troquem informações através de conexões socket;

TOleStream – Permite a leitura e escrita de informação disponibilizada através de um objeto OLE.

Como um TStream pode armazenar qualquer tipo de informações, isso significa que até mesmo objetos da VCL podem ser contidos dentro dele. Um uso clássico para TStream é a possibilidade de se armazenar em um meio físico o estado de um dado objeto, isso é uma forma de persistência de objetos. Por exemplo, imagine um TListBox onde os usuários adicionam informações em tempo de execução. Depois essas informações precisam ser salvas em algum lugar para que sejam recuperadas em um uso subsequente. Para muitos a solução seria ter uma tabela em um banco de dados ou criar até mesmo um arquivo texto e lançar ali o conteúdo desse objeto. Não são ideias ruins, mas elas custam um pouco mais de código. Uma solução alternativa é fazer uso do TStream.

O próprio Delphi utiliza esse tipo de persistência. Toda vez que um projeto é salvo ele grava em disco o conteúdo de seus objetos, assim quando o projeto é recarregado o Delphi coloca tudo de volta como estava. No código a seguir é possível ver como um objeto TListBox pode ser salvo na forma de arquivo:

```
procedure TForm1.FormClose(
  Sender: TObject;
  var Action: TCloseAction);
var
  strm : TFileStream;
begin
  strm := TFileStream.Create('MinhaLista.DAT',
  fmCreate);
  strm.WriteComponent(ListBox1);
  strm.Free;
end;
```

Já no código seguinte o valor é recuperado:

```
procedure TForm1.FormCreate(
  Sender: TObject);
var
  strm : TFileStream;
begin
  if FileExists('MinhaLista.DAT') then
  begin
    strm := TFileStream.Create('MyList.DAT',
    fmOpenRead);
    strm.ReadComponent(ListBox1);
    strm.Free;
  end;
end;
```

Nada complexo. A funcionalidade toda está a cargo dos métodos ReadComponent e WriteComponent. Respectivamente um lê as propriedades do objeto passado e o

segundo grava no objeto passado o conteúdo contido no arquivo MinhaLista.DAT.

Compressão de arquivos

No primeiro projeto usando o TZipMaster será escolhido um arquivo qualquer, pode ser uma foto, texto ou programa para que seja comprimido. É ideal que seja um texto ou documento do Office, visto que alguns tipos de arquivos por já estarem em um formato comprimido, não podem ser muito comprimidos pelo zipmaster. Por exemplo, um arquivo mp3 teria pouquíssima diferença de tamanho comprimido em formato zip.

Listagem 1. Primeiro exemplo: compactando arquivo

```
procedure TfrmComprimir.btComprimirClick
 (Sender: TObject);
begin
  with TOpenDialog.Create(nil) do
  try
    Filter := 'Todos os arquivos (*.*)|*.*';
    Title := 'Escolha um arquivo para comprimir';
    if Execute(Handle) then
    begin
      ZipArq.FSpecArgs.AddStrings(Files);
    end
    else
    begin
      ZipArq.FSpecArgs.Clear;
      exit;
    end;
  finally
    free;
  end;
  with TSaveDialog.Create(nil) do
  try
    Filter := 'Arquivos ZIP (*.zip)|*.zip';
    Title := 'Salvando o arquivo ZIP';
    if Execute(Handle) then
    begin
      ZipArq.ZipFileName := FileName;
      ZipArq.Add;
    end
    else
    begin
      ZipArq.FSpecArgs.Clear;
      exit;
    end;
  finally
    Free;
  end;
end;
```

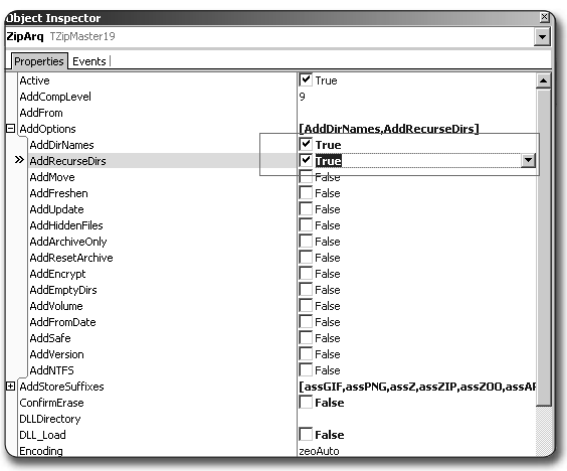


Figura 1. Opções de compressão

O primeiro exemplo consiste de um formulário com um componente TZipMaster e um TButton. O evento OnClick mostrado na **Listagem 1** cria em tempo de execução um OpenDialog para que seja selecionado um arquivo a ser comprimido. Depois destrói o mesmo e cria um SaveDialog para ser escolhido onde será salvo o arquivo .zip. A propriedade ZipArq.FSpecArgs do ZipArc, do tipo TStrings, guarda a lista dos arquivos a serem compactados. Por enquanto trabalharemos apenas com um arquivo. Caso o usuário clique em cancelar em qualquer uma das caixas de diálogo, Abrir ou Salvar, a operação será abortada com um exit, mas antes disso a lista de arquivos será limpa.

A compressão é feita apenas se o usuário clicar em Abrir e depois em Salvar. Por isso a parte mais importante do processamento está dentro de um IF Execute(handle). Passar o handle do formulário para o método execute significa que a caixa de diálogo Abrir ou Salvar será modal em relação ao formulário. Isso significa que o formulário não poderá ser clicado enquanto os diálogos de Abrir e Salvar estiverem abertos. No Abrir foi adicionado o atributo Files ao FSpecArgs usando o método AddStrings. Nesse caso Files terá apenas um arquivo. No salvar foi usado o método Add para compactar o arquivo. Com essa pequena implementação já é possível compactar um arquivo.

Agora vamos considerar um pequeno ajuste para que vários arquivos possam ser salvos dentro de um único arquivo .zip. Os caminhos absolutos desses arquivos também serão salvos, para que eles possam ser extraídos com suas respectivas pastas, mesmo que extraídos diretamente na raiz do sistema.

Abaixo do título temos a linha Options := Options+ [ofAllowMultiSelect], responsável por adicionar ao OpenDialog a opção de permitir que múltiplos arquivos sejam selecionados. A linha ZipArq.AddOptions := ZipArq.AddOptions + [AddDirNames, AddRecurseDirs], colocada logo acima da chamada ao método Add, fará com que sejam salvas as informações de diretórios dos arquivos a serem adicionados. A opção AddDirNames adiciona os nomes dos diretórios e a opção AddRecurseDirs adiciona arquivos e diretórios recursivamente. Ambas opções podem ser alteradas no próprio Object Inspector, conforme mostra a **Figura 1**.

Por enquanto não será apresentado como adicionar diretórios completos, pois o OpenDialog não permite isso. Para testar a alteração, executamos o programa para compactar todos os arquivos constantes na pasta do exemplo. Isso inclui os .pas, .dfm e .exe. Abrindo com o 7Zip o arquivo gerado, a primeira pasta que pode ser visualizada é realmente a primeira pasta da cadeia com o endereço completo do arquivo, nesse caso Documentos, conforme pode ser visto na **Figura 2**. Se navegar na estrutura, o diretório onde estão os arquivos selecionados será visualizado.

A propriedade AddCompLevel é um inteiro que pode controlar a taxa de compressão dos arquivos. O padrão é o máximo, nove, mas podemos diminuir se quisermos aumentar a velocidade de processamento em troca de arquivos finais um pouco maiores. Embora os arquivos testados, juntos, tenham 1,21mb de tamanho, o arquivo comprimido tem apenas 603 kb no nível 9.

Compressão de diretórios inteiros

Para comprimir diretórios é necessário abrir mão do OpenDialog e usar algo que possa abrir um diretório inteiro, isso pode ser feito com o componente TDirectoryOu-

tline. O objetivo é simples: o usuário irá selecionar um diretório e ao clicar em Comprimir o diretório selecionado será compactado e salvo todo em um único arquivo, preservando a estrutura de diretórios e subdiretórios interna. O código para isso pode ser visto na **Listagem 2**.

Dividindo em volumes

Arquivos ou diretórios muito grandes podem ser comprimidos dividindo-se em volumes pequenos o suficiente para serem armazenados em CDs, disquetes ou em tamanho viável para se fazer upload ou mandar por e-mail. A opção MaxVolumeSizeKb define o tamanho em kb que um arquivo deve ter, neste caso 1024kb é usado para dividir o arquivo em partes de 1mb. Existe outra propriedade chamada WriteOptions que pode ser um conjunto de várias opções de escrita. Essas podem ser ativadas pelo Object Inspector, ou via código. A opção considerada é a zwoDiskSpan, que força a divisão do arquivo no tamanho especificado em MaxVolumeSizeKb mesmo que haja espaço suficiente em disco. Caso contrário o arquivo só será dividido se não houver espaço suficiente no disco removível, onde o programa solicitará a inserção de outro disco. A opção spCompatName da propriedade SpanOptions faz com que todas as partes do arquivo tenham o mesmo nome, mas com as extensões variando numericamente, como z01,z02 e assim por diante. A **Listagem 3** mostra o código com estas opções.

Listando arquivos dentro do Zip

Um TListBox ou TMemo pode ser usado para listar os arquivos dentro de um zip. A ideia do exemplo da **Figura 3** é abrir um arquivo zipado e já listar seus arquivos. Se um arquivo estiver selecionado na List Box ele é extraído, caso contrário tudo é considerado.

O componente TZipMaster funciona como se fosse uma lista. Ele tem uma propriedade count que armazena o número total de arquivos dentro do zip, é possível indexar o ZipMaster com os colchetes [] como se fosse a propriedade items de um Listbox. Cada item armazena propriedades dos arquivos internos, como nome, tamanho real, tamanho comprimido e outras. A **Listagem 4** (código do botão Abrir) varre esses itens com um laço for e adiciona no Listbox os nomes dos arquivos.

Para extrair os arquivos o ZipMaster precisa de um diretório base onde são descompactados seus arquivos. O próprio TDirectoryOutline é usado para isso, como mostra o código a seguir (botão Extrair):

```
procedure TfrmComprimir.bt
ExtrairClick(Sender: TObject);
begin
  ZipArq.ExtrBaseDir :=
  diretorio.Directory;
  ZipArq.Extract;
end;
```

Para testar basta abrir um arquivo zip qualquer que tenha vários diretórios ou caminhos completos dentro. Ao extraí-los pode-se perceber que a antiga estrutura de diretórios não é mais obedecida. Para manter a estrutura de diretórios original deve-se colocar a opção ExtrOptions := ExtrOptions + [ExtrDirNames].

O código da **Listagem 5** é uma alteração da **Listagem 4** e mostra como varrer as linhas selecionadas do ListBox e extrair somente essas. Existe uma outra opção de ExtrOptions importante, que não será mostrada no exemplo, que é ExtrOverWrite. Ela permite que arquivos já existentes com o mesmo nome no destino sejam sobrescritos. Sem essa opção os arquivos não serão extraídos.

Programa instalador

Assim como o WinZip e WinRar podem criar arquivos compactos auto-extraíveis e assim criar mini instaladores, o ZipMaster também pode. Isso é demonstrado através do projeto de exemplo chamado "InstallMaker" (**Figura 4**). Um outro formulário é usado para selecionar os arquivos que serão adicionados no projeto (**Figura 5**).

Listagem 2. Compressão de diretórios

```
procedure TfrmComprimir.btComprimirClick
  (Sender: TObject);
begin
  ZipArq.FSpecArgs.Add(diretorio.Directory+'\');
  With TSaveDialog.Create(nil) do
  try
    Filter := 'Arquivos ZIP (*.zip)|*.zip';
    Title := 'Salvando o arquivo ZIP';
    if Execute(Handle) then
    begin
      ZipArq.ZipFileName := FileName;
      ZipArq.AddOptions := ZipArq.AddOptions
      + [AddDirNames, AddRecurseDirs];
      ZipArq.Add;
    end
    else
    begin
      ZipArq.FSpecArgs.Clear;
      exit;
    end;
  finally
    Free;
  end;
end;
```

Figura 2. Abrindo o arquivo com o 7Zip

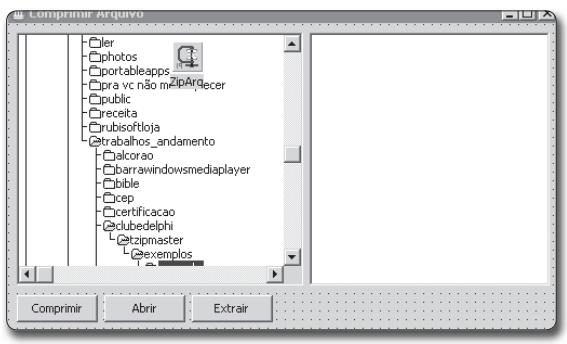


Figura 3. Botões Abrir e Extrair

Figura 4. Instalador

Listagem 3. Compressão de diretórios

```
procedure TfrmComprimir.btComprimirClick
 (Sender: TObject);
begin
  ZipArq.FSpecArgs.Add(diretorio.Directory+'\');
  with TSaveDialog.Create(nil) do
  try
    Filter := 'Arquivos ZIP(*.zip)|*.zip';
    Title := 'Salvando o arquivo ZIP';
    DefaultExt := '.zip';
    if Execute(Handle) then
    begin
      ZipArq.ZipFileName := FileName;
      ZipArq.AddOptions := ZipArq.AddOptions
     + [AddDirNames, AddRecurseDirs];
      ZipArq.MaxVolumeSizeKb :=
       1024; //1024 kb = 1mb
      ZipArq.WriteOptions := [zwoDiskSpan];
      ZipArq.SpanOptions := [spCompatName];
      ZipArq.Add;
    end
    else
    begin
      ZipArq.FSpecArgs.Clear;
      exit;
    end;
  finally
    Free;
  end;
end;
```

Listagem 4. Listando arquivos

```
procedure TfrmComprimir.btAbrirClick
(Sender: TObject);
var
  i: integer;
begin
  with TOpenDialog.Create(nil) do
  try
    Filter := 'Arquivos ZIP (*.zip)|*.zip';
    Title := 'Salvando o arquivo ZIP';
    DefaultExt := '.zip';
    if  Execute(handle) then
    begin
      ZipArq.ZipFileName := FileName;
      for i := 0 to ZipArq.Count - 1 do
      begin
        lstArquivos.Items.Add(ZipArq[i].FileName);
      end;
    end;
  finally
    Free;
  end;
end;
```

Listagem 5. Extração de arquivos selecionados

```
procedure TfrmComprimir.btExtrairClick
 (Sender: TObject);
var
  i: integer;
begin
  ZipArq.FSpecArgs.Clear;
  for I := 0 to lstArquivos.Count - 1 do
  begin
    if lstArquivos.Selected[i] then
      ZipArq.FSpecArgs.Add(lstArquivos.Items[i]);
  end;
  ZipArq.ExtrBaseDir := diretorio.Directory;
  ZipArq.ExtrOptions := ZipArq.ExtrOptions
  + [ExtrDirNames];
  ZipArq.Extract;
end;
```

O botão para adicionar diretório (">>>") inclui o diretório inteiro selecionado no DirectoryListBox. O botão para adicionar arquivos (">>") inclui os arquivos selecionados. O botão remover ("<<") retira os arquivos da lista. O de Ok adiciona os arquivos da lista ao FSpecArgs do zip. A **Listagem 6** mostra a unit completa.

Uma particularidade da **Listagem 6** é que nos botões para adicionar diretórios ou arquivos é verificado se o mesmo já não existe. Isso porque colocar duas vezes o mesmo arquivo no zip ocasionará um erro. Já para excluir os selecionados o método DeleteSelected do TListBox é usado.

A **Listagem 7** possui o código da unit principal, que cria um programa de arquivos compactados auto-extraíveis (self extract) ou SFX. Esse tipo de arquivo é um executável que contém os outros arquivos comprimidos dentro dele. Esse tipo de arquivo é muito usado para criar instaladores simples e Patches. O arquivo foi criado sem DiskSpan ou divisão em volumes de tamanho específico. Também não considera o endereço completo dos arquivos, mas apenas seus caminhos relativos ao diretório que foi adicionado, e recursivamente seus subdiretórios.

Só de adicionar os arquivos.bin da pasta RES no diretório do executável e colocar a extensão do arquivo como .exe já causará a criação de um arquivo SFX. Para testar o instalador pode-se criar um programa "cobaia", que não faz nada, tem apenas um formulário vazio. Ao rodar o programa basta preencher corretamente os campos, como mostra o exemplo da **Figura 6**.

O instalador pode ser mais direto ao ponto que execute o programa instalado no final do processo de instalação. Isso é possível ativando-se as propriedades soAutoRun para o instalador começar a rodar automaticamente. A opção soNoSuccessMsg suprimi a mensagem

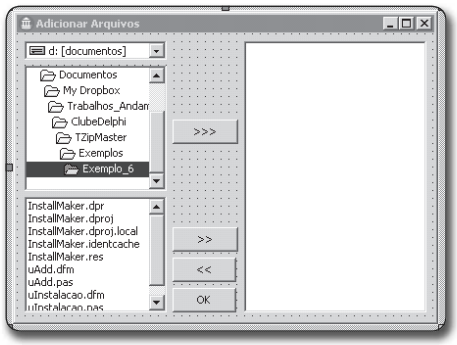


Figura 5. Formulário para adicionar aquivos

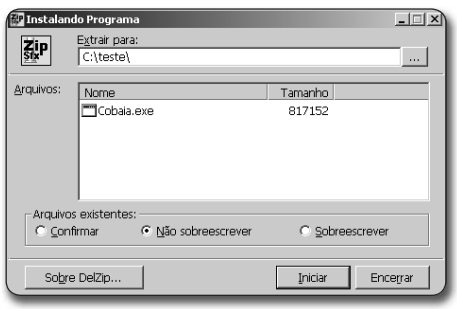


Figura 6. Janela do instalador

de sucesso no final do processo, enquanto soCreateEmptyDirs cria os diretórios de destino caso não existam (a **Figura 7** mostra o Object Inspector com essas opções ativadas).

Para que o programa instalado execute automaticamente depois do processo de instalação, a linha de comando deve ser informada como o endereço de destino onde o programa vai estar, nesse caso "c:\teste\cobaia.exe". Também pode ser colocado aqui arquivos .bat, arquivos de registro para importação ou qualquer outro comando ou chamada a programa que exista na máquina de destino.

"Zipando" um stream

Muitos de nós criamos cadastros de clientes onde colocamos a foto do cliente. Ou cadastro de produtos com a foto do mesmo. Isso também é útil em sistemas de controle de portarias ou emissão de carteirinhas de associado ou crachás. Muitas pessoas salvam a foto em forma de stream em um campo blob no banco de dados, para evitar que o arquivo se perca com o tempo. Isso pode fazer o banco de dados crescer muito rápido, por isso seria interessante se pudéssemos compactar antes de guardar no blob.

Listagem 6. Unit uAdd

```
unit uAdd;

interface

uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
 Forms, Dialogs, StdCtrls, FileCtrl;

type
 TfrmAdd = class(TForm)
   flbArquivos: TFileListBox;
   dlbDiretorio: TDirectoryListBox;
   cmbUnidade: TDriveComboBox;
   lstArquivos: TListBox;
   btAdicionarDir: TButton;
   btRemover: TButton;
   btOk: TButton;
   btAdicionarArquivo: TButton;
   procedure btOkClick(Sender: TObject);
   procedure FormCreate(Sender: TObject);
   procedure btAdicionarDirClick(Sender: TObject);
   procedure btAdicionarArquivoClick(Sender: TObject);
   procedure btRemoverClick(Sender: TObject);
 private
   { Private declarations }
 public
   { Public declarations }
 end;

var
 frmAdd: TfrmAdd;

implementation

uses uInstalacao;

{$R *.dfm}

procedure TfrmAdd.btAdicionarArquivoClick(Sender: TObject);
var
 i: integer;
begin
 for I := 0 to flbArquivos.Count - 1 do
 begin
  if (flbArquivos.Selected[i]) and (lstArquivos.Items.IndexOf
  (flbArquivos.Items[i])<0) then
    lstArquivos.Items.Add(flbArquivos.Items[i]);
 end;
end;

procedure TfrmAdd.btAdicionarDirClick(Sender: TObject);
var
 sDir: string;
begin
 sDir := dlbDiretorio.Directory+'\';
 if lstArquivos.Items.IndexOf(sDir) < 0 then
   lstArquivos.Items.Add(dlbDiretorio.Directory+'\');
end;

procedure TfrmAdd.btOkClick(Sender: TObject);
begin
 frmInstalador.zip.FSpecArgs.Clear;
 frmInstalador.zip.FSpecArgs.AddStrings(lstArquivos.Items);
 Close;
end;

procedure TfrmAdd.btRemoverClick(Sender: TObject);
begin
 lstArquivos.DeleteSelected;
end;

procedure TfrmAdd.FormCreate(Sender: TObject);
begin
 lstArquivos.Clear;
 lstArquivos.Items.AddStrings(frmInstalador.zip.FSpecArgs);
end;

end.
```

Listagem 7. Unit uInstalacao

```
unit uInstalacao;

interface

uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
 Forms,
 Dialogs, StdCtrls, ZipMstr19;

type
 TfrmInstalador = class(TForm)
   zip: TZipMaster19;
   edTitulo: TEdit;
   Label1: TLabel;
   edDiretorio: TEdit;
   Label2: TLabel;
   edComando: TEdit;
   Label3: TLabel;
   edMensagem: TEdit;
   Label4: TLabel;
   btCriar: TButton;
   btAdicionar: TButton;
   procedure btAdicionarClick(Sender: TObject);
   procedure btCriarClick(Sender: TObject);
 private
   { Private declarations }
 public
   { Public declarations }
 end;

var
 frmInstalador: TfrmInstalador;

implementation

uses uAdd;

{$R *.dfm}

procedure TfrmInstalador.btAdicionarClick(Sender: TObject);
begin
 with TfrmAdd.Create(nil) do
 try
   ShowModal;
 finally
   Release;
```

Continuação: Listagem 7. Unit uInstalacao

```
  end;
end;

procedure TfrmInstalador.btCriarClick(Sender: TObject);
begin
  with TSaveDialog.Create(nil) do
  try
    DefaultExt := '*.exe';
    Filter := 'Arquivos Executáveis (*.exe)|*.exe';
    if Execute(Handle) then
    begin
      ShowMessage('Adicionando arquivos: ' + #13#10
      + zip.FSpecArgs.Text);
      zip.ZipFileName := FileName;
      zip.AddOptions := zip.AddOptions + [AddRecurseDirs];
      zip.SFXCaption := edTitulo.Text;
      zip.SFXCommandLine := edComando.Text;
      zip.SFXDefaultDir := edDiretorio.Text;
      zip.SFXMessage := edMensagem.Text;
      zip.Add;
    end;
  finally
    Free;
  end;
end;

end.
```

Listagem 8. SQL de criação do banco

```
create table clientes (
CodCli integer not null primary key,
Nome varchar(50),
foto blob
)

create generator gnclientes

CREATE TRIGGER "INSCLIENTES" FOR
"CLIENTES"
ACTIVE BEFORE INSERT POSITION 0
AS
   BEGIN
      NEW.codcli = GEN_ID(gnclientes, 1);
   END
```

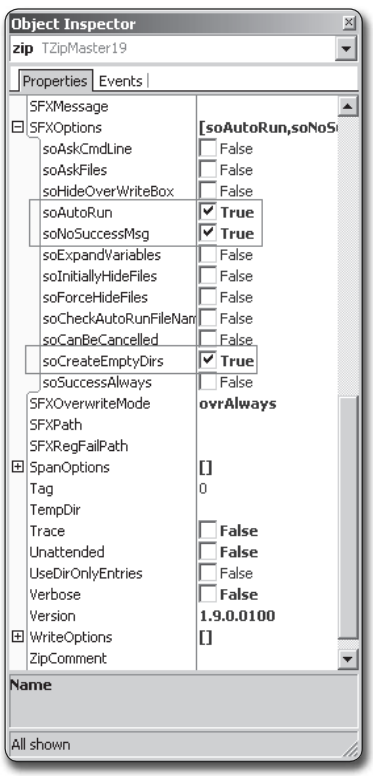


Figura 7. O arquivo no seu destino

O exemplo a seguir é um cadastro básico no Firebird (um banco com uma única tabela), com um programa que acessa esse cadastro. Será mostrado como salvar a figura integralmente e depois como salvar a figura compactada com um zipstream. A **Listagem 8** mostra os comandos SQL para criar a tabela de clientes, o generator e a trigger para controlar o incremento da chave primária.

Não serão usados os mecanismos normais para gravar ou apresentar imagens no formulário, o processo manual dará uma flexibilidade maior. A **Figura 8** mostra o formulário e a **Listagem 9** mostra o código da unit inteira. O form contém os componentes básicos de conexão do dbExpress. Um TImage com as propriedades proportional e stretch como true fará com que sempre fique quadrado e que diminua a imagem para que possa ser exibida nele. A imagem gravada no campo blob é lida com a ajuda de um stream. Ao se navegar pelo ClientDataSet a imagem é carregada para o TImage . O programa considera apenas imagens bmp por serem maiores em tamanho, tornando evidentes os benefícios da compressão. Além disso, elas são mais simples de se trabalhar, tornando o exemplo mais claro e didático.

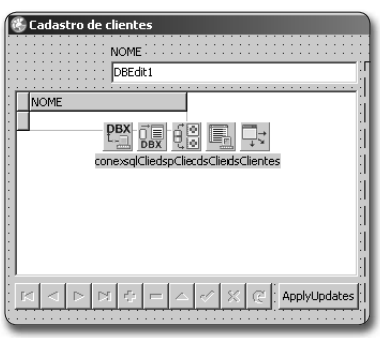


Figura 8. Formulário de cadastro com foto

O projeto tem duas procedures importantes. O evento OnClick do img TImage abre um TOpenDialog para escolher e abrir um arquivo .bmp e depois esse arquivo será aberto por um filestream. O campo blob armazena esse filestream e servirá também para exibir a imagem. Para isso é preciso verificar antes qual o estado do ClientDataSet, se ele não estiver em edição ou inserção é preciso colocá-lo em edição ou não permitir a alteração. A procedure ExibirFoto é chamada no evento OnChange do dsClientes. Assim, sempre que o ClientDataSet for aberto ou navegado, a procedure será executada. Essa procedure tem o papel de fazer o processo inverso: obter o stream do campo blob e exibí-lo no TImage.

A **Listagem 9** grava e lê as imagens do banco de dados, mas ainda não faz a compressão. Logo no evento OnCreate do formulário o parâmetro Database do SQLConnection é configurado como cadastro.fdb, localizado no mesmo diretório do executável. Também consta código para abrir o ClientDataSet e o ApplyUpdates. O método ExibirFoto verificará primeiramente se o ClientDataSet está ativo, se o campo foto não é nulo, ou seja, se tem foto cadastrada, e se o ClientDataSet não está em modo de edição. Só assim a foto estará pronta para ser exibida.

A **Listagem 10** é um cópia modificada do código da **Listagem 9**. Nesta cópia

são armazenadas imagens compactadas no banco de dados para economizar espaço, por isso existe um componente TZipMaster ("zip"). O banco de dados cadastro.fdb usado é uma cópia limpa do banco.

As diferenças entre as **Listagens 9** e **10** encontram-se apenas nas procedures ExibirFoto e imgImagemClick. No evento OnClick do TImage existe agora um FileStream e dois MemoryStreams. O FileStream lerá o arquivo. O primeiro MemoryStream (strMemoria) copia os dados do stream da imagem para si. Isso porque os métodos do zip usados não trabalham com FileStreams, mas apenas com MemoryStreams. O stream da imagem é carregado com o arquivo aberto no TOpenDialog.

O strZipado (também um TMemoryStream) não precisa ser criado porque uma instância é retornada pelo método AddStreamToStream. Esse método recebe o strMemoria, comprime-o e passa o resultado compactado para o strZipado. Não é preciso criar nenhum arquivo zip intermediário: todo o processo ocorre na memória.

Nota

Uma cópia do banco de dados em branco é feita, para gravar as imagens sem compressão e outro para gravar com compressão, permitindo a comparação dos tamanhos dos arquivos.

Listagem 9. Colocando imagens integrais no banco de dados

```
unit uCadastro;

interface

uses
  Windows, Messages, SysUtils, Variants,
 Classes, Graphics, Controls, Forms,
  Dialogs, WideStrings, DBXFirebird,
  FMTBcd, DBClient, Provider, DB, SqlExpr,
  StdCtrls, Mask, DBCtrls, ExtCtrls, Grids, DBGrids;

type
  TfrmCadastro = class(TForm)
    conexao: TSQLConnection;
    sqlClientes: TSQLDataSet;
    dspClientes: TDataSetProvider;
    cdsClientes: TClientDataSet;
    DBNavigator1: TDBNavigator;
    Label1: TLabel;
    DBEdit1: TDBEdit;
    dsClientes: TDataSource;
    btApply: TButton;
    DBGrid1: TDBGrid;
    cdsClientesNOME: TStringField;
    cdsClientesFOTO: TBlobField;
    imgImagem: TImage;
    procedure btApplyClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure imgImagemClick(Sender: TObject);
    procedure dsClientesDataChange(Sender: TObject; Field: TField);
  private
    { Private declarations }
  public
    { Public declarations }
    procedure ExibirFoto;
  end;

var
  frmCadastro: TfrmCadastro;

implementation

{$R *.dfm}

procedure TfrmCadastro.btApplyClick(Sender: TObject);
begin
  cdsClientes.ApplyUpdates(0);
end;

procedure TfrmCadastro.dsClientesDataChange(Sender: TObject; Field:
TField);
begin
  ExibirFoto;
end;

procedure TfrmCadastro.ExibirFoto;
var
  strImagem: TMemoryStream;
begin
  if cdsClientes.Active and
    not cdsClientesFOTO.IsNull and
    not (cdsClientes.State in [dsInsert, dsEdit]) then
  begin
    strImagem := TMemoryStream.Create;
    cdsClientesFOTO.SaveToStream(strImagem);
    strImagem.Position := 0;
    imgImagem.Picture.Bitmap.LoadFromStream(strImagem);
    strImagem.Free;
  end;
end;

procedure TfrmCadastro.FormCreate(Sender: TObject);
begin
    conexao.Params.Add('Database=' +
    ExtractFilePath(Application.ExeName)+'\CADASTRO.FDB');
  cdsClientes.Open;
end;

procedure TfrmCadastro.imgImagemClick(Sender: TObject);
var
  strImagem: TFileStream;
begin
  if not (cdsClientes.State in [dsInsert, dsEdit]) then
    cdsClientes.Edit;
  with TOpenDialog.Create(nil) do
  try
    Filter := 'Arquivos bmp|*.bmp';
    if Execute(Handle) then
    begin
      strImagem := TFileStream.Create(FileName, fmOpenRead);
      cdsClientesFOTO.LoadFromStream(strImagem);
      strImagem.Position := 0;
      imgImagem.Picture.Bitmap.LoadFromStream(strImagem);
    end;
  finally
    Free;
  end;
end;

end.
```

Listagem 10. Colocando imagens compactas no banco de dados

```
unit uCadastro;

interface

uses
  Windows, Messages, SysUtils, Variants,
  Classes, Graphics, Controls, Forms,
  Dialogs, WideStrings, DBXFirebird,
  FMTBcd, DBClient, Provider, DB, SqlExpr,
  StdCtrls, Mask, DBCtrls, ExtCtrls, Grids,
  DBGrids, jpeg, ZipMstr19;

type
  TfrmCadastro = class(TForm)
    conexao: TSQLConnection;
    sqlClientes: TSQLDataSet;
    dspClientes: TDataSetProvider;
    cdsClientes: TClientDataSet;
    DBNavigator1: TDBNavigator;
    Label1: TLabel;
    DBEdit1: TDBEdit;
    dsClientes: TDataSource;
    btApply: TButton;
    DBGrid1: TDBGrid;
    cdsClientesNOME: TStringField;
    cdsClientesFOTO: TBlobField;
    imgImagem: TImage;
    zip: TZipMaster19;
    procedure btApplyClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure imgImagemClick(Sender: TObject);
    procedure dsClientesDataChange(Sender: TObject; Field: TField);
  private
    { Private declarations }
  public
    { Public declarations }
    procedure ExibirFoto;
  end;

var
  frmCadastro: TfrmCadastro;

implementation

{$R *.dfm}

procedure TfrmCadastro.btApplyClick(Sender: TObject);
begin
  cdsClientes.ApplyUpdates(0);
end;

procedure TfrmCadastro.dsClientesDataChange(Sender: TObject; Field:
TField);
begin
  ExibirFoto;
end;

procedure TfrmCadastro.ExibirFoto;
var
  strMemoria: TMemoryStream;
  strZipado: TMemoryStream;
begin
  if cdsClientes.Active and
    not cdsClientesFOTO.IsNull and
    not (cdsClientes.State in [dsInsert, dsEdit]) then
  begin
    zip.Clear;
    strZipado := TMemoryStream.Create;
    cdsClientesFOTO.SaveToStream(strZipado);
    strZipado.Position := 0;
    strMemoria := zip.ExtractStreamToStream(strZipado, strZipado.Size);
    strMemoria.Position := 0;
    imgImagem.Picture.Bitmap.LoadFromStream(strMemoria);
    strZipado.Free;
  end;
end;

procedure TfrmCadastro.FormCreate(Sender: TObject);
begin
        conexao.Params.Add('Database=' +
        ExtractFilePath(Application.ExeName)+'\CADASTRO.FDB');
  cdsClientes.Open;
end;

procedure TfrmCadastro.imgImagemClick(Sender: TObject);
var
  strImagem: TFileStream;
  strMemoria: TMemoryStream;
  strZipado: TMemoryStream;
begin
  if not (cdsClientes.State in [dsInsert, dsEdit]) then
    cdsClientes.Edit;
  with TOpenDialog.Create(nil) do
  try
    Filter := 'Arquivos bmp|*.bmp';
    if Execute(Handle) then
    begin
      zip.Clear;
      strImagem := TFileStream.Create(FileName, fmOpenRead);
      strMemoria := TMemoryStream.Create;
      strMemoria.LoadFromStream(strImagem);
      strZipado := zip.AddStreamToStream(strMemoria);
      //para mostrar a diferença de tamanhos
      //ShowMessage('Arquivo: ' + inttostr(strImagem.Size));
      //ShowMessage('Zipado: ' + inttostr(strZipado.Size));
      cdsClientesFOTO.LoadFromStream(strZipado);
      strImagem.Position := 0;
      imgImagem.Picture.Bitmap.LoadFromStream(strImagem);
      strImagem.Free;
      strMemoria.Free;
    end;
  finally
    Free;
  end;
end;

end.
```

As linhas comentadas são ShowMessages para mostrar a diferença de tamanho (propriedade size dos streams). O LoadFromStream(strZipado) carrega para dentro do campo blob o conteúdo já zipado. Com o LoadFromStream(strImagem) o TImage é carregado a partir do bloco de memória integral e não do comprimido. No final strImagem e strMemoria são destruídos, nunca deve-se destruir strZipado.

Na procedure ExibirFoto dois TMemoryStreams são criados, um para o stream zipado e outro para o integral. O primeiro comando deve ser zip.Clear para limpar qualquer informação que o zip possa ter a respeito de arquivos ou em seu stream interno. Isso porque essa operação se repetirá várias vezes conforme é feita a navegação no ClientDataSet. Uma outra alternativa seria a cada chamada desse método criar uma instância de TZipMaster e depois destruí-la logo após o uso.

Com os comandos strZipado := TMemoryStream.Create e cdsClientesFOTO.SaveToStream(strZipado) o Memory Stream que armazena os dados comprimidos é criado, nele é carregado o conteúdo do campo blob. As linhas strZipado.Position := 0 e strMemoria := zip.ExtractStreamTo Stream(strZipado, strZipado.Size) posicionam o strZipado em seu início. O método ExtractStreamToStream coloca o conteúdo zipado para o strMemoria.

Repare que nessa operação ocorre o contrário da operação de compressão e gravação da foto. O strZipado é criado e destruído, mas o strMemoria não pode ser criado porque ele é resultado do método ExtractStreamToStream e é ao mesmo tempo uma referência à própria propriedade ZipStream do componente zip, como comentado anteriormente. Isso tudo quer dizer que a propriedade ZipStream será um stream com conteúdo descompactado quando alguma operação de extração for executada, ou será o conteúdo compactado quando a operação de Adição/Compressão for feita. Esse stream mudará todo momento conforme o método que for executado.

A linha strMemoria.Position := 0 posiciona o recém criado stream strMemoria em seu início e a linha LoadFromStream(strMemoria) carrega o strMemoria para a TImage. A imagem aparecerá normalmente porque o strMemoria agora é o arquivo "natural". Neste método pode-se destruir o strZipado, mas nunca strMemoria, ou seja, o contrário do evento OnClick do TImage.

Ao rodar os programas das **Listagens 9** e **10**, pode-se adicionar sistematicamente as mesmas imagens com os mesmos nomes. Também é preciso escolher imagens grandes, como fotos em formato bmp. As fotos usadas nesse exemplo encontram-se nos arquivos desta edição. Depois de dar Apply Updates e fechar os dois aplicativos é possível comparar o tamanho dos bancos de dados do Firebird. Com isso aprendemos a comprimir streams de fotos e salvá-los onde quisermos sem termos que passar por um arquivo zip.

Programa de backup

O último exemplo mostrado na **Figura 9** e **Listagens 12** e **13** utiliza as técnicas já apresentadas de uso do TZipMaster para criar um sistema de backup para o Firebird. A aplicação usa o programa gback.exe que vem junto com o Firebird para fazer backup de um banco de dados e depois compactar o fbk gerado com TZipMaster para ficar menor ainda. Ao rodar o programa e clicar no botão "Fazer Backup" algumas vezes, será criada a pasta Backups onde serão salvos vários backups nela, com os horários no nome.

Nota

Sempre que copiamos, movemos ou carregamos dados de um stream para outro, é preciso colocar a posição do stream para zero (Position := 0), caso contrário o stream será copiado de seu final, e não haverá nenhum byte.

Listagem 11. Função ExecutaEspera

```
unit uBackup;

interface

uses
  Windows, Messages, SysUtils, Variants,
  Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ZipMstr19, IBServices;

type
  TfrmBackup = class(TForm)
    zip: TZipMaster19;
    btBackup: TButton;
    procedure btBackupClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    procedure Backup;
  end;

var
  frmBackup: TfrmBackup;


function ExecutaEspera(
  const Arquivo, Parametros: string;
  const EstadoJanela: Word): boolean;

implementation

{$R *.dfm}

{ TfrmBackup }

function ExecutaEspera(
  const Arquivo, Parametros: string;
  const EstadoJanela: Word): boolean;
var
  InfoInicializacao: TStartupInfo;
  InfoProcesso: TProcessInformation;
  Comando: string;
begin
  //Coloca o nome do arquivo entre aspas.
  //Isto é necessário se o arquivo estiver em
  // pastas com espaços nos nomes
  Comando := '"' + Arquivo + '" ' + Parametros;
  FillChar(InfoInicializacao, SizeOf(InfoInicializacao), #0);
  with InfoInicializacao do
  begin
    cb := SizeOf(InfoInicializacao);
    dwFlags := STARTF_USESHOWWINDOW;
    //SW_Hide ou 0 esconde, SW_Shownormal ou 1 mostra
    wShowWindow := EstadoJanela;
  end;
  Result := CreateProcess(nil, PChar(Comando), nil, nil, false,
    CREATE_NEW_CONSOLE or NORMAL_PRIORITY_CLASS, nil,
    PChar(ExtractFilePath(Arquivo)), InfoInicializacao, InfoProcesso);
  //verifica se conseguiu criar o processo
  if Result then
  begin
    //aguarda até o término do processo.
    //Só passará à linha de código abaixo
    //quando terminar
    WaitForSingleObject(InfoProcesso.hProcess, INFINITE);
    //fechando os handles que manipulam os processos
    CloseHandle(InfoProcesso.hProcess);
    CloseHandle(InfoProcesso.hThread);
  end;
end;
```

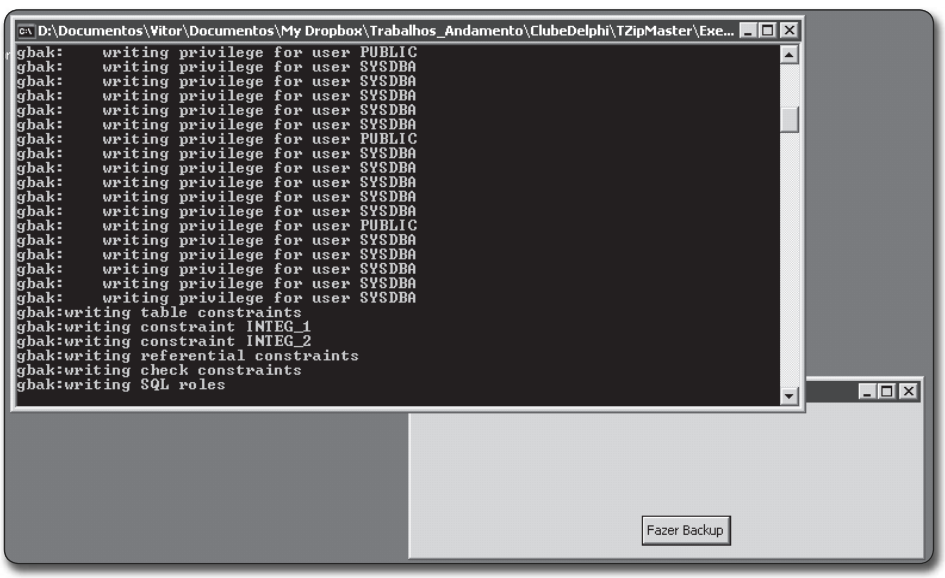


Figura 9. Programa em execução

É necessário usar uma função (**Listagem 11**) para executar um programa externo e esperar pelo término de sua execução, isso é feito através da API do Windows. Um maior detalhe sobre o funcionamento dessa função não será abordado porque não faz parte do escopo desse artigo, no entanto existem comentários nos comandos menos óbvios para que possa se ter um mínimo de entendimento de como funciona. Na seção interface consta o cabeçalho da função, pois essa é uma função estática global.

A função ExecutaEspera faz uma chamada ao método CreateProcess da API do Windows para criar um processo. Ela recebe como parâmetro uma estrutura do tipo TStartupInfo que contém informações sobre a inicialização do processo. Se a criação do processo for bem sucedida o método WaitForSingleObject é usado para executar o processo e esperar até que acabe.

Quando tudo ficar pronto será possível ver a janela de console do programa gbak fazendo o backup do banco de dados Firebird. Isso é necessário porque não podemos fazer uma cópia ou compressão de um arquivo fdb com usuários conectados e operando transações, com o risco de corromper o banco.

A **Listagem 12** mostra o código de backup, que simplesmente faz uma chamada a essa procedure. Para fazer backup com o gbak são necessárias algumas informações importantes: o nome ou ip do servidor do banco de dados, o nome e endereço do arquivo do banco, login do Firebird, password e o nome do arquivo de destino

Listagem 12. Método Backup

```
procedure TfrmBackup.Backup;
var
  arqBak: string;
  arqZip: string;
  backupString: string;
  tempData: string;
  pathexe: string;
  gbak: string;
  login: string;
  senha: string;
  servidor: string;
  banco: string;
begin
  //caminhos de arquivos e configurações
  pathexe := ExtractFilePath(Application.ExeName);
  gbak := pathexe + 'gbak.exe';
  login := 'sysdba';
  senha := 'masterkey';
  servidor := 'localhost';
  //nomes de arquivos colocandos entre aspas duplas
  banco := '"'+pathexe+'cadastro.fdb'+'"';
  //cria o diretório Backups se este não existir
  ForceDirectories(pathexe + 'Backups');
  //define o nome de acordo com a data, no formato
  //Ano_mes_dia_hora_minuto_segundo
  tempData := pathexe + 'Backups\' + 'Backup' +
     formatdatetime('_YYYY_MM_DD_HH_NN_SS', Now);
  //define o nome do arquivo fbk e zip
  //nomes de arquivos colocandos entre aspas duplas
  arqBak :=  '"'+tempData+'.fbk'+'"';
  //o nome do arquivo zip não precisa de aspas pois
  //o zipmaster lida corretamente com ele
  arqZip :=  tempData+'.zip';
  //comando a ser usado com o gbak
  backupString :=
     ' -V -B ' + servidor + ':' + banco +
     ' -l -user ' + login +
     ' -pas ' + senha + '  ' + arqBak;
  //execução do gbak com
  //SW_SHOWNORMAL para mostrar a janela do programa
  ExecutaEspera(gbak, backupString, SW_SHOWNORMAL);
  //compressão do fbk para ficar ainda menor
  with TZipMaster19.Create(nil) do
  try
    Active := true;
    DLLDirectory := pathExe;
    AddCompLevel := 9;
    try
      FSpecArgs.Add(arqBak);
      ZipFileName := arqZip;
      Add;
    except
      on e: Exception do
      begin
        MessageBox(0,
          PChar('Erro ao criar arquivo compactado!' +
            #13#10 + e.Message),
          'erro de backup', MB_ICONERROR);
      end;
    end;
    finally
      //SuccessCnt é uma propriedade do TZipMaster
      //que conta o total de arquivos
      //adicionados com sucesso. No nossa caso é apenas um: arqBak
      if FileExists(arqZip) and (SuccessCnt > 0) then
      begin
        //se tudo correu bem podemos deletar o fbk,
        //senão devemos guarda-lo
        DeleteFile(arqBak);
        MessageBox(0,
          PChar('Backup concluido com êxito'),
          'Backup', MB_ICONINFORMATION);
      end
      else
      begin
        MessageBox(0,
          PChar('Erro ao criar arquivo compactado! O fbk foi mantido.' ),
            'erro de backup', MB_ICONERROR);
      end;
      //não esquecer de destruir o TZipMaster no final
      Free;
    end;
end;
```

do backup, o .fbk. Essas informações poderiam vir de uma caixa de diálogo, de um arquivo .ini ou de um arquivo .xml. Como não é o escopo desse artigo ver essas implementações, estes dados estão no próprio código fonte ("hard coded").

O caminho do banco e consequentemente dos arquivos fbk e zip é o caminho do próprio executável, por isso existe a variável pathexe que terá o valor de ExtractFilePath(Application.ExeName). Os arquivos fbk e fdb são colocados com aspas, pois o gbak é uma aplicação console, se tiver espaços em branco em nomes longos de arquivos e diretórios eles serão interpretados erroneamente. Os arquivos de backup fbk e zip são colocados na pasta Backups. ForceDirectories assegura que ela exista ou que seja criada caso não exista. A função FormatDateTime('_YYYY_MM_DD_HH_NN_SS', Now) obtém a string com a data no formato inverso. Isso é útil para os arquivos na pasta backups poderem ser facilmente visualizados um abaixo do outro, mesmo em ordem alfabética.

Tanto o fbk como o zip terão as datas em seus nomes. A variável backupString recebe uma string com os parâmetros para o programa gbak. Os parâmetros são endereço do servidor e do arquivo fdb, login, senha, opção –v para ter mais informações sobre o processo de backup no console (verbosity), -b para especificar que a operação é um backup, -l para ignorar transações no limbo (isso elimina transações não "comitadas" e lixo, diminuindo o tamanho do banco), login e senha. No final de tudo vai o arquivo .fbk. O método ExecutaEspera será executado passando-se o endereço do programa gbak e os parâmetros, além da flag SW_SHOWNORMAL para poder ver o console do gbak.

Depois disso uma instância de um TZip Master é criada, o arquivo fbk é adicionado e finalmente é criado um .zip, como já mostrado nas primeiras listagens deste artigo. Se houver erros durante o processo de compressão é porque algum erro ocorreu durante a criação do fbk e o mesmo não existe. O gbak é executado com a opção –y <endereço> para criação de um log, assim é possível ver o que ocorreu. Se houver outros erros durante

a compressão ou no final do processo, o arquivo .zip não vai existir e será exibido um alerta dizendo que o backup não foi concluído corretamente. Se tudo correr bem será mostrada a mensagem 'Backup concluído com êxito'.

Conclusão

ZipMaster não é a única alternativa para compressão de arquivos pelo Delphi. Você pode usar também as APIs do 7Zip se preferir algo GPL, ou pode usar RAR, ACE, ARJ e tantas outras bibliotecas e algoritmos. Escolhemos o formato ZIP simplesmente pela sua portabilidade e popularidade, afinal quase todos os programas de compressão ou backup podem ler ou abrir arquivos ZIP. Agora você pode comprimir arquivos, criar pequenos instaladores e atualizações, além de programas de backup, tanto de arquivos como de bancos de dados. No diretório de exemplos do ZipMaster temos muitos exemplos interessantes e úteis, incluindo um quase clone do WinZip.

Vitor Luiz Rubio
vitorrubio@gmail.com - http://www.vitorrubio.blogspot.com
Tecnólogo em Processamento de Dados pela FATEC-SP e MCP. Atua como Analista de Sistemas Sênior na Editora Revista dos Tribunais. Programa com Delphi desde a versão 3 e com .Net desde a versão 1.1.

Blogs do autor
http://www.vitorrubio.blogspot.com/
http://twitter.com/vitorrubio

TZipMaster
http://www.delphizip.org/

WinZip
http://www.corel.com/servlet/Satellite/br/pt/Product/1207834997200#versionTabview=tab1&tabview=tab0

WinRar
http://www.rarlab.com/

7Zip
http://www.7-zip.org/

JEDI
http://www.delphi-jedi.org/

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Cursos Online

Assinatura Java Magazine PLUS
Mais conteúdo JAVA por muito menos!

A Revista **Java Magazine** oferece para seus assinantes uma série de Cursos Online de alto padrão de qualidade .

Conheça abaixo o curso já disponível:
www.devmedia.com.br/curso/javamagazine

Desenvolvendo um cadastro de notas fiscais e produtos com Struts 2

A sua melhor opção de aprendizagem!
Assine a **Java Magazine** e Comece já seu treinamento!
www.devmedia.com.br/assine

DevMedia Mais Informações: www.devmedia.com.br/central - Tel.: 21 3382-5038

Subversion
Controle de versão no Delphi XE

Rafael Stavarengo

Resumo DevMan

De que trata o artigo:

O artigo ensina usar o novo cliente para Subversion que já vem integrado com o novo IDE do Delphi XE. Se você não está familiarizado com sistemas controladores de versão, não se preocupe, pois o artigo também explica o que é e como funciona este tipo de sistemas usando o Subversion como exemplo, que atualmente é o controlador de versão mais popular.

Para que serve:

O novo IDE do Delphi XE vem cheio de recursos e um deles é a integração com o Subversion. Você aprenderá como usar esta nova ferramenta para aumentar sua produtividade e desempenho.

Em que situação o tema é útil:

Seja em equipe ou sozinho, trabalhar em um bom projeto exige boa administração. Os sistemas controladores de versão facilitam o gerenciamento do projeto inclusive sobre o controle de acesso, definindo quem pode fazer o que com os arquivos.

Subversion:

Conheça um pouco mais do novo IDE XE do Delphi aprendendo a trabalhar com projetos integrados a um sistema controlador de versão e entenda como eles funcionam. Através de um exemplo o artigo mostra como gerenciar projetos usando o Subversion, desde a criação do repositório até as funções básicas do cotidiano, como Update e Commit. Não se preocupe se não souber o significado destes novos termos, pois também aprenderá o que eles significam.

O artigo apresenta como usar o IDE do novo Delphi XE integrado com projetos gerenciados pelo controlador de versão mais popular do mercado: o Subversion (SVN). Ao nos familiarizarmos com os controladores de versão vamos entender a filosofia destes programas e para que eles realmente foram inventados. Através de exemplos práticos o leitor entenderá o que é um repositório, serão criados e não apenas usados os que já existem. Serão abordadas as diferenças entre os sistemas existentes e explicado porque o Subversion vem se destacando entre eles.

Quando se fala em consumir os recursos do Subversion usando o Windows fica difícil não falar sobre o TortoiseSVN, para gerenciar repositórios do Subversion. Vamos também apresentá-lo.

VCS? Mas não era sobre SVN?

Version Control System (VCS), ou sistema controlador de versões, é uma especificação de programas de computador usados para gerenciar as alterações feitas em outros arquivos. Estes sistemas são geralmente usados por desenvolvedores de software que precisam de um histórico do que foi alterado em seus projetos, porém pode-se usar um controlador de versão em qualquer tipo de projeto uma vez que estes sistemas são capazes de gerenciar alterações em qualquer tipo de arquivo.

VCS não é um programa de computador mais sim uma especificação que define o que um programa precisa para ser considerado um controlador de versão. Em resumo, se um programa for capaz de controlar as alterações de um arquivo ele pode ser considerado um VCS.

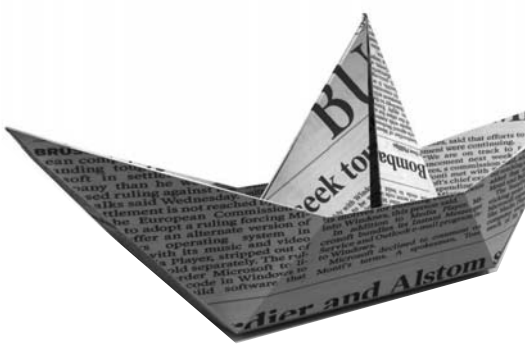
Hoje existem controladores de versão com licença gratuita e paga. Os VCS gratuitos mais populares são CVS, GIT e o Subversion (SVN), sendo este último um novo CVS – logo saberá por quê. Dentre as implementações de VCS com licença paga podemos citar o ClearCase da IBM e o SourceSafe da Microsoft.

Vantagens

A primeira grande vantagem de se usar um VCS é ter acesso a todas as versões de cada arquivo. Isto significa que o VCS guarda um histórico de todas as alterações dos arquivos e o mais interessante é que é possível comparar as diferenças entre as versões, ou seja, podemos ver qual a diferença (o que foi alterado) entre uma versão e outra.

Os controladores de versão normalmente possuem algum tipo de controle de usuários. Além de ver o que foi alterado, ainda saberemos quem foi que fez a alteração e porque a alteração foi feita, pois sempre que uma nova versão é criada o usuário pode criar uma observação dizendo o porquê de tal modificação. Normalmente esta observação não é obrigatória.

O controle de usuários não serve apenas para saber quem fez o quê. Com os controladores de versão você também é capaz de



dizer quem tem acesso ao quê. Por exemplo, é possível definir que o Joãozinho, estagiário, que entrou na empresa semana passada só poderá alterar os arquivos da interface gráfica do projeto, porém, o Manuel que é o chefe da equipe tem acesso irrestrito a todos os arquivos.

Com um programa de versionamento é possível marcar um ponto específico na vida do projeto e sempre que necessário voltar até aquele ponto. Por exemplo, podemos criar uma marca indicando o momento na vida do projeto em que ele estava estável (sem bugs), e continuar fazendo as alterações. Se precisarmos da última versão estável basta recuperar os arquivos do ponto que foi marcado. Os pontos criados para marcar um momento específico são chamados de Tag.

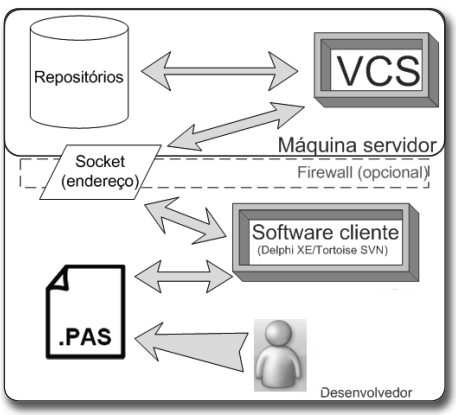


Figura 1. Esquema físico

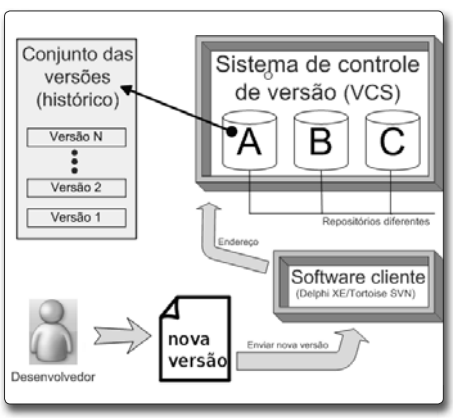


Figura 2. Esquema lógico

Nota do DevMan

Não podemos confundir VCS com CVS (Version Control System com Concurrent Versions System). Como já foi dito VCS é uma especificação do que um programa precisa para ser considerado um controlador de versões. Por outro lado o CVS é uma das implementações do VCS, ou seja, um programa controlador de versões.

Quando temos Tags, logo podemos criar Branch. Com os controladores de versão é possível ramificar um projeto, ou seja, dividi-lo em várias linhas de desenvolvimento diferentes, que mais tarde serão todas mescladas. Assim quando precisar fazer uma alteração tão grande que pode até atrapalhar o trabalho dos outros programadores, podemos dividir o programa criando outra ramificação do projeto onde somente determinada pessoa vai trabalhar. Quando estiver tudo certo, mesclamos as alterações feitas na ramificação com a ramificação principal onde todos estão trabalhando.

Falando em trabalhar em equipe, um grande problema é o conflito de edição. Um problema sério que acontecia sempre que dois programadores tentavam alterar o mesmo arquivo. Eu disse que "acontecia", pois com os controladores de versão este problema está quase extinto, já que vários programadores podem trabalhar no mesmo arquivo ao mesmo tempo. Quando terminarem o que estão fazendo, as alterações feitas serão mescladas e ninguém perderá uma linha de código.

Como funciona um VCS

Cada controlador de versão gerencia as versões dos arquivos de maneira diferente, mas todas elas possuem um repositório. O repositório é o local onde ficam os arquivos controlados e o histórico das alterações destes arquivos. O formato do repositório varia entre os programas existentes: pode ser tanto uma pasta dentro do HD, um banco de dados ou qualquer outra forma de armazenamento onde os arquivos e os históricos possam ser armazenados.

Este repositório terá seu próprio endereço e deve ficar em um local de comum acesso por toda a equipe (normalmente no servidor da empresa). O formato do endereço varia podendo ser uma URL, um endereço do sistema de arquivo local ou outro qualquer. Na **Figura 1** vemos o processo.

A **Figura 1** mostra como funciona a comunicação entre a máquina do desenvolvedor (máquina cliente) e o servidor (máquina com o repositório). Normalmente a máquina cliente terá um programa cliente e através deste programa o desenvolvedor terá acesso ao repositório dentro do servidor. Este programa cliente pode ser, por exemplo, o novo Delphi XE, pois ele já vem com um suporte para trabalhar com o Subversion. Claro que para ter acesso ao repositório o desenvolvedor precisa conhecer o endereço do repositório. É através do software cliente que o desenvolvedor tem acesso a todas as informações, como por exemplo, o histórico de versões. O que cada desenvolvedor pode ou não fazer dentro do repositório pode ser controlado através dos níveis de acesso que os controladores de versão possuem.

Também é necessário entender o esquema lógico do processo, como mostra a **Figura 2**.

A **Figura 2** mostra como funciona o esquema lógico do VCS. A parte chamada "Sistema de controle de versão" representa o que seria, por exemplo, o Subversion rodando no servidor. Repare que podem existir mais de um repositório no mesmo servidor e cada repositório controla seus respectivos arquivos, assim é possível ter um repositório para cada projeto. Se preferir também pode-se colocar todos os projetos em um único repositório, nada impede isto. Na **Figura 2**, um desenvolvedor fez alguma alteração em um dos arquivos do Repositório "A" e através do programa cliente o desenvolvedor está submetendo as alterações deste arquivo de volta para o repositório. Isto cria uma nova versão do arquivo que ele alterou.

CVS ou SVN

Concurrent Versions System (CVS), ou Sistema de Versões Concorrentes foi por muitos anos o programa controlador de versões dominante, pois além do fato de ser open-source ele atendia a maioria das necessidades de um trabalho corporativo. Na verdade ele ainda atende uma vez que não deixou de existir. Como o Concurrent Versions System permitia o acesso tanto pelo sistema de arquivos local como também pela rede, era ideal para que as equipes pudessem trabalhar sem que os

integrantes da equipe estivessem no mesmo local físico.

Mas o CVS também tinha seus problemas e corrigi-los poderia ser muito penoso. O Subversion ou SVN surge para ser a nova versão do CVS. Atualmente o SVN é o controlador de versão favorito dos desenvolvedores de programas open-source, até por que o próprio SVN é um programa open-source.

Subversion e repositórios

O Delphi XE vem com uma ferramenta embutida que atua como um programa cliente do Subversion, por isto funções como criação de repositórios não podem ser executadas pelo IDE, já que esta é uma função do servidor do Subversion. Portanto antes de usar o Delphi XE para trabalhar com o Subversion, é preciso entender um pouco do TortoiseSVN.

O TortoiseSVN funciona tanto como aplicação cliente quanto como aplicação servidora para o Subversion. Quando o assunto é Subversion no Windows ele é um dos programas mais difundidos, apesar de existirem outros. O TortoiseSVN é open-source e o link para download está na sessão Links ao final do artigo, portanto é necessário baixá-lo e instalá-lo. Depois de instalado o Subversion se integra ao Windows Explorer, oferece opções específicas através do menu de contexto (clique do botão direito do mouse).

A primeira coisa a se fazer para utilizar SVN é criar um repositório onde ficarão os arquivos que serão controlados, juntamente com o histórico de versões. Para criar o repositório deve-se escolher uma pasta vazia, clicar com o botão direito do mouse sobre ela, apontar sobre a opção TortoiseSVN e escolher Create repository here (**Figura 3**).

Dentro da pasta do repositório serão criados alguns arquivos e diretórios, mas nenhum deles é relevante, pois não é necessário entender como as coisas funcionam dentro do repositório (fisicamente), é necessário saber que é no repositório que ficam todos os arquivos de projetos, mesmo que não possam ser vistos. Mas se não é possível entender como os arquivos estão organizados dentro do repositório, como acessá-los? Vejamos a seguir.

Acesso aos arquivos do repositório

Os repositórios podem ser acessados por um endereço remoto (HTTP) ou pelo sistema de arquivos local. Mas independente de como o repositório será acessado, criá-lo será sempre igual ao que acabamos de ver. Para podermos acessar o repositório e ver o que tem dentro dele usando o TortoiseSVN, acessamos sua opção Repo-browser para indicar qual o repositório deve ser acessado, como mostra a **Figura 4**.

No tópico anterior o repositório foi criado na mesma máquina de onde será acessado, ou seja, de forma local. Este acesso também pode ser remoto. Para demonstrar isto, antes de navegar para repositório recém criado, vamos acessar o repositório do projeto ACBr. Na caixa de diálogo URL informamos o endereço https://acbr.svn.sourceforge.net/svnroot/acbr/trunk para que as informações sejam carregadas, como aparece na **Figura 5**.

O endereço para o repositório do projeto ACBr começa com o protocolo HTTP, isto porque o repositório está em algum lugar na internet e foi necessária uma conexão remota. Para acessar um repositório usando uma conexão local é necessário usar o protocolo de conexão FILE. Para isso informamos o caminho "file:///" seguido do caminho da pasta do repositório com as barras invertidas. Por exemplo: Se temos o repositório na pasta "D:\Meus documentos\repositorio" seu caminho então é "file:///D:/Meus documentos/repositorio". O conteúdo do repositório criado será mostrado, porém ele está vazio. Como fazer para adicionar arquivos ao repositório?

Import

Para que um arquivo tenha suas alterações controladas, é necessário que ele esteja dentro de seu repositório. O ato de adicionar os primeiros arquivos em um novo repositório é chamado de Import.

É muito simples fazer Import com o TortoiseSVN, uma vez selecionado(s) o(s) arquivo(s), acessamos a opção Import e indicamos para qual repositório o arquivo deve ir (**Figura 6**).

Em resumo, "Import" é a ação de adicionar os primeiros arquivos em um novo repositório e somente os arquivos que estiverem dentro do repositório serão controlados pelo Subversion - obviamente os arquivos precisam existir antes de serem adicionados ao repositório. Esses arquivos podem acessados pelo Repo-browser.

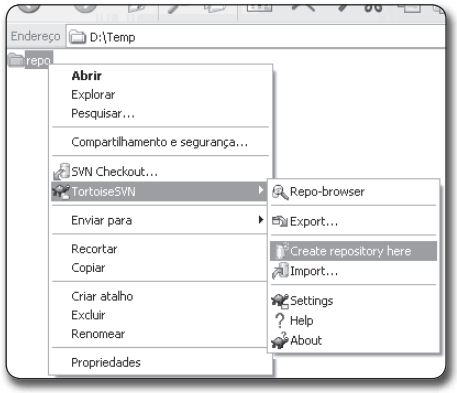


Figura 3. Criando um repositório

Nota do DevMan

O ACBr é um conjunto de componentes com licença open-source para Delphi VCL ou CLX que permite acesso direto a equipamentos como Impressoras Fiscais, Gavetas de Dinheiro entre outras coisas, sem que precise ficar estudando sobre as DLLs criadas pelos fabricantes destes equipamentos. Talvez o componente mais popular do ACBr seja o ACBrNFe usado para trabalhar com nota fiscal eletrônica. Mais informações sobre o projeto ACBrNFe na edição 108 da ClubeDelphi.

SVN no Delphi XE

O novo IDE do Delphi XE traz ferramentas integradas para trabalhar com projetos no SVN, trazendo o máximo de produtividade enquanto o versionamento do seu código é feito tanto individualmente quanto em equipe. Para demonstrar seu uso criamos um novo projeto, configurado como na **Figura 7** e **Tabela 1**. Nele vamos simular várias alterações e seu devido controle.

Esta é a primeira versão dos arquivos do projeto. Seguindo a lógica anteriormente citada, é preciso agora incluir esse projeto ao repositório. Dentro do IDE temos opções para isso. Ao acessarmos o Project Manager vemos o item Add to version control, como mostra a **Figura 8**.

A diferença de usar o SVN pelo IDE do Delphi em relação ao TortoiseSVN é que o IDE simplifica ainda mais o seu trabalho.

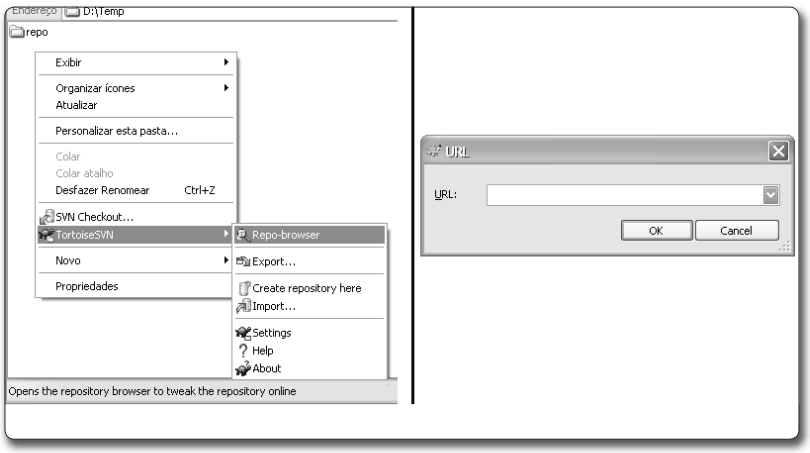


Figura 4. Navegando nos arquivos de um repositório

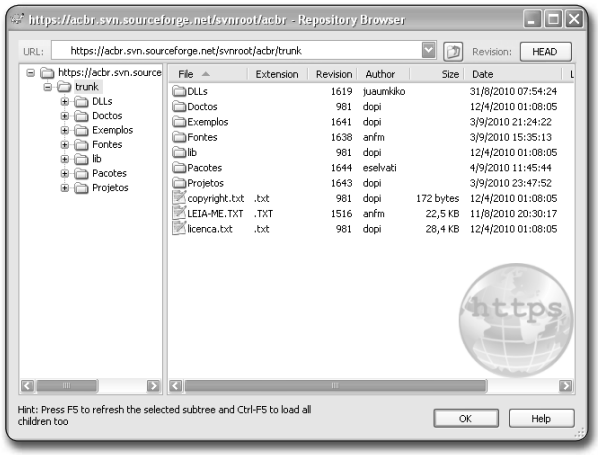


Figura 5. O arquivos do repositório do projeto ACBr

Componente	Propriedade	Valor
TEdit	Name	Edit1
TEdit	Name	Edit2
TComboBox	Name	ComboOperacao
	Items	mais menos vezes dividido por
	ItemIndex	0
	Style	csDropDownList
TLabel	Name	Label1
	Caption	=
TLabel	Name	LabelResultado
	Caption	result
TForm	Name	Form1
	Caption	Form1

Tabela 1. Componentes do Form1

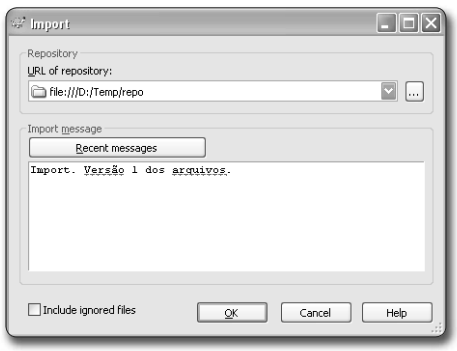


Figura 6. Fazendo Import

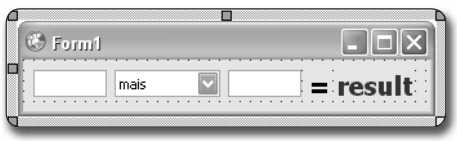


Figura 7. Layout do Form1

Um exemplo disto pode ser visto na **Figura 8**. Os arquivos Calculadora.dproj.local e Calculadora.identcache além de outras pastas e arquivos ocultos criados automaticamente pelo Delphi, não aparecem na lista de arquivos a serem importados. Somente os arquivos realmente necessários são adicionados ao repositório. O TortoiseSVN não saberia diferenciar os arquivos temporários dos arquivos realmente necessários, apesar de ser possível ignorá-los manualmente.

Checkout

Uma vez que os arquivos do projeto estão dentro do repositório, os originais não são mais necessários e podem ser excluídos da pasta. Atenção! O repositório não deve ser excluído, pois é nele que estão os arquivos originais, apagamos somente os arquivos da pasta onde está o projeto e não da pasta onde está o repositório.

Qualquer pessoa de qualquer lugar poderá acessar o repositório, desde que ela saiba o endereço e tenha privilégio de acesso. Quando estas pessoas quiserem alterar este projeto, elas pegam uma cópia da última versão dos arquivos existentes no repositório. Isto significa que não trabalhamos com os arquivos diretamente dentro do repositório, ao invés disto guardamos uma cópia localmente da última versão existente no repositório e usamos esta cópia para fazer as alterações que precisamos. A pasta onde guardamos a cópia destes arquivos é chamada de Working Copy ou Cópia de Trabalho.

Para criar uma Working Copy é necessário realizar o Checkout dos arquivos. Para fazer isso no TortoiseSVN, em uma nova pasta, deve-se acessar o menu de contexto e utilizar a opção Checkout, onde pode-se informar qual repositório copiar, como mostra a **Figura 9**.

Nota

A opção Add to version control é utilizada para inclusão de projetos, arquivos, a um repositório e não para envio de alterações realizadas.

O Delphi XE também pode fazer o Checkout dos arquivos. No menu File temos a opções Open from version control. Assim como acontece no TortoiseSVN, uma janela chamada Checkout será exibida, com a diferença que nessa pode-se informar o caminho da pasta onde o working copy será criado (**Figura 10**). A vantagem de se fazer Checkout pelo Delphi XE aparece agora, pois depois que o Delphi termina a

operação, uma lista com todos os projetos encontrados durante o Checkout, aparece para que possamos escolher qual abrir (**Figura 11**).

Submetendo as alterações

O processo que envia atualizações no working copy para o repositório é chamado de Commit Na **Listagem 1** foi feita uma alteração na versão original.

Não existe nada de especial no código da **Listagem 1**, até porque o objetivo não é aprender programação mais sim aprender a submeter alterações feitas em um projeto controlado pelo SVN.

Entre as linhas 1 e 15, está a implementação do método Calcular. Ele converte os valores dos componentes Edit1 e Edit2, depois aplica a operação que o usuário selecionou no componente ComboOperacao. Em seguida entre as linhas 16 e 27 estão as implementações do evento OnChange de cada um dos componentes que invocará o método Calcular sempre que o usuário mudar alguma coisa em um dos dois TEdit ou no TComboBox.

Para submeter estas alterações (executar o Commit), usando o Delphi XE, acessamos a opção Subversion do Project Manager e nela temos Commit. Em Commit temos From Repository Root. Assim a tela para envio das alterações é exibida (**Figura 12**), já ignorando todos os arquivos desnecessários, como aqueles que ficam na pasta oculta chamada __history.

Abaixo da lista de arquivos alterados está um espaço para a observação sobre o que foi mudado. Esta observação não é obrigatória, mas faz toda a diferença para quem precisa encontrar uma versão específica ou para quem está querendo saber se determinado erro já foi corrigido, podendo ser nós mesmos. Esta observação deve ser feita com muitos detalhes descrevendo principalmente o motivo da alteração e não somente qual foi a alteração. Para submeter alterações ao repositório clicamos sobre o botão Commit.

Listagem 1. Alterando o projeto

```
01  procedure TForm1.Calcular;
02  var
03    v1, v2: Integer;
04    Resultado: Real;
05  begin
06    v1 := StrToInt(Edit1.Text);
07    v2 := StrToInt(Edit2.Text);
08    case ComboOperacao.ItemIndex of
09      0: Resultado := v1 + v2;
10      1: Resultado := v1 - v2;
11      2: Resultado := v1 * v2;
12      3: Resultado := v1 / v2;
13    end;
14    LabelResultado.Caption := FloatToStr(Resultado);
15  end;

16  procedure TForm1.ComboOperacaoChange(Sender: TObject);
17  begin
18    Calcular;
19  end;
20  procedure TForm1.Edit1Change(Sender: TObject);
21  begin
22    Calcular;
23  end;
24  procedure TForm1.Edit2Change(Sender: TObject);
25  begin
26    Calcular;
27  end;
```

Mais alterações... mais commits

Antes de partir para outro assunto, vamos fazer outra alteração no projeto Calculadora. Um olhar mais atento deve ter notado um

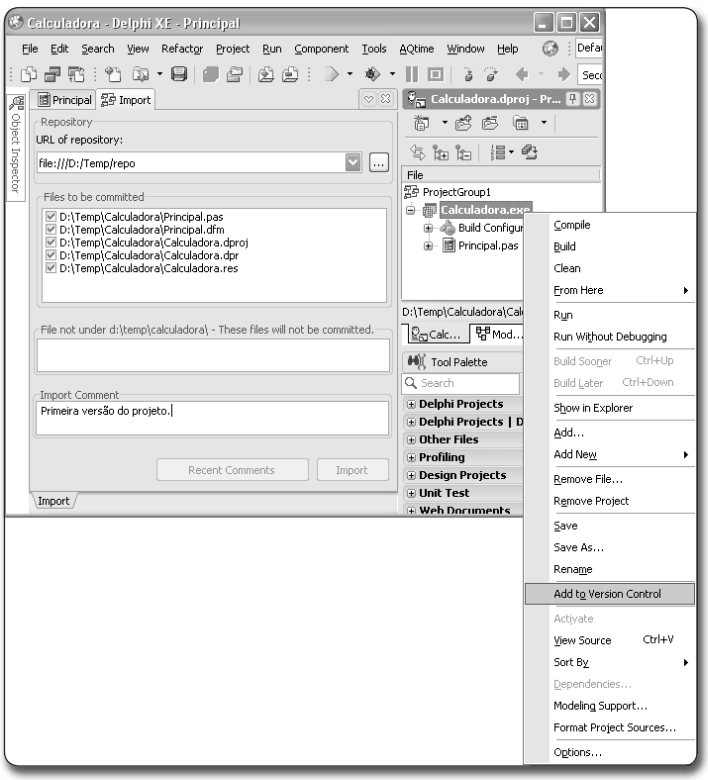


Figura 8. Adicionando um projeto ao repositório

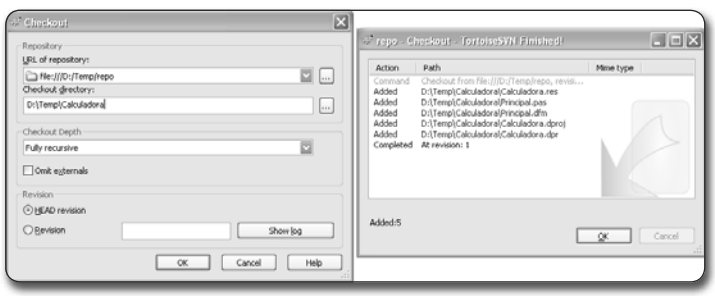


Figura 9. Checkout com TortoiseSVN

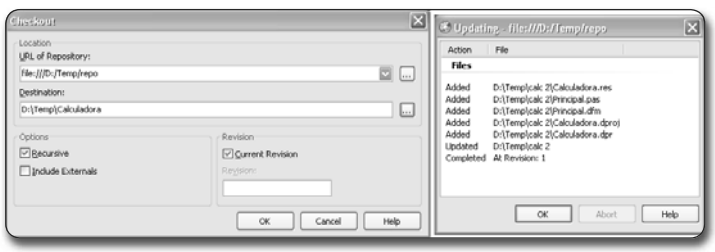


Figura 10. Checkout com Delphi XE

bug na aplicação. Acontece que o método Calcular tenta converter os valores dos TEdit para Integer, como pode ser visto nas linhas 6 e 7 da **Listagem 1**, porém, esta conversão não é segura e vai gerar exceções se o conteúdo dos TEdits não puder ser convertido. Para corrigir este erro será necessário substituir o que aparece nas linhas 6 e 7 da **Listagem 1** pelo que é mostrado a seguir:

```
...
6           v1 := StrToIntDef(Edit1.Text, 0);
7           v2 := StrToIntDef(Edit2.Text, 0);
...
```

Para fazer o Commit com o TortoiseSVN, abrimos o Windows Explorer e navegamos até o working copy que provavelmente terá um ícone com um símbolo de exclamação vermelho indicando que alguma coisa foi alterada lá dentro. Acessando o menu de contexto da pasta working copy, escolhemos a opção SVN Commit. Assim como no Delphi XE aparecerá uma janela com a lista dos arquivos alterados e um espaço para descrição do que foi alterado. Se efetuarmos um duplo clique sobre um arquivo, as alterações feitas nele serão exibidas.

Nota do DevMan

Depois de feita a correção mostrada, se o Windows Explorer for aberto na pasta working copy, os ícones dos arquivos vão aparecer alterados. Estes novos ícones são aplicados graças ao TortoiseSVN. Quando um arquivo tem um ícone com um símbolo verde significa que ainda não sofreu alterações. Um arquivo cujo ícone tem um símbolo vermelho significa que foi modificado. Existem outros símbolos além destes. Às vezes pode acontecer de o Windows mostrar um ícone incorreto por alguns instantes, como por exemplo, mostrar um símbolo verde quando deveria ser vermelho. Para solucionar isso basta pressionar a tecla F5 para que seja feito um refresh (atualização) da exibição dos arquivos.

Nota

Ao fazer Commit com o TortoiseSVN, não podemos nos esquecer de ignorar os arquivos temporários gerados pelo Delphi, se eles ainda não estiverem ignorados.

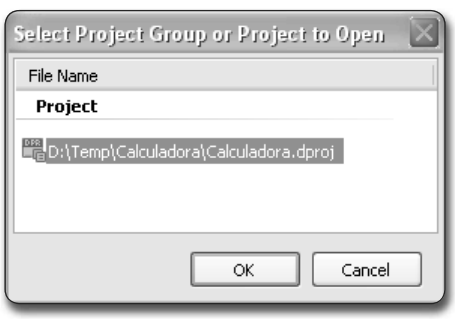


Figura 11. Projetos encontrados durante o checkout com Delphi XE

Analisando o histórico

Observar o histórico de versões é uma operação comum, principalmente quando procuramos por algo específico, o motivo de algo ter sido alterado, saber o que exatamente foi alterado para corrigir um bug ou implementar uma nova função, saber o que os outros programadores mudaram no projeto ou simplesmente para lembrar o que foi feito ontem, antes de irmos para casa.

Mais uma vez o novo Delphi XE sai na frente em relação ao TortoiseSVN. Com o XE sempre temos disponível o histórico do arquivo que está sendo editado na mesma guia onde o arquivo está sendo editado. Para entender melhor acompanhe a explicação usando a **Figura 13**.

Em qualquer unit que estiver em edição, existirá logo abaixo do editor a opção History (número 1 na **Figura 13**). Ao clicar nele temos acesso tanto ao histórico local quanto ao histórico do repositório. O histórico local é feito automaticamente pelo Delphi e este histórico não tem nada a ver com o histórico do repositório. O Delphi cria uma nova versão de histórico local cada vez que o arquivo for salvo. Este histórico criado pela IDE fica dentro da pasta oculta chamada __history no mesmo local onde o projeto está salvo.

Os históricos do repositório possuem um ícone azul e estão marcados pelo número 2 na **Figura 13**. Já o histórico local tem um ícone verde e estão marcados pelo número 3. Facilmente podemos acessar a observação de uma versão ou comparar a diferença entre as elas clicando respectivamente nas opções Information e Differences que estão destacadas com o número 4, também na **Figura 13**.

Para acessar o histórico através do TortoiseSVN, acessamos o menu de contexto da pasta working Copy e selecionamos o item Show log.

Update

Update é a ação de atualizar o working copy para que os arquivos reflitam a última versão existente no repositório. Na verdade o Update faz um pouco mais do que simplesmente atualizar para a última versão, pois no caso de existir alterações no seu working copy, elas não serão perdidas,

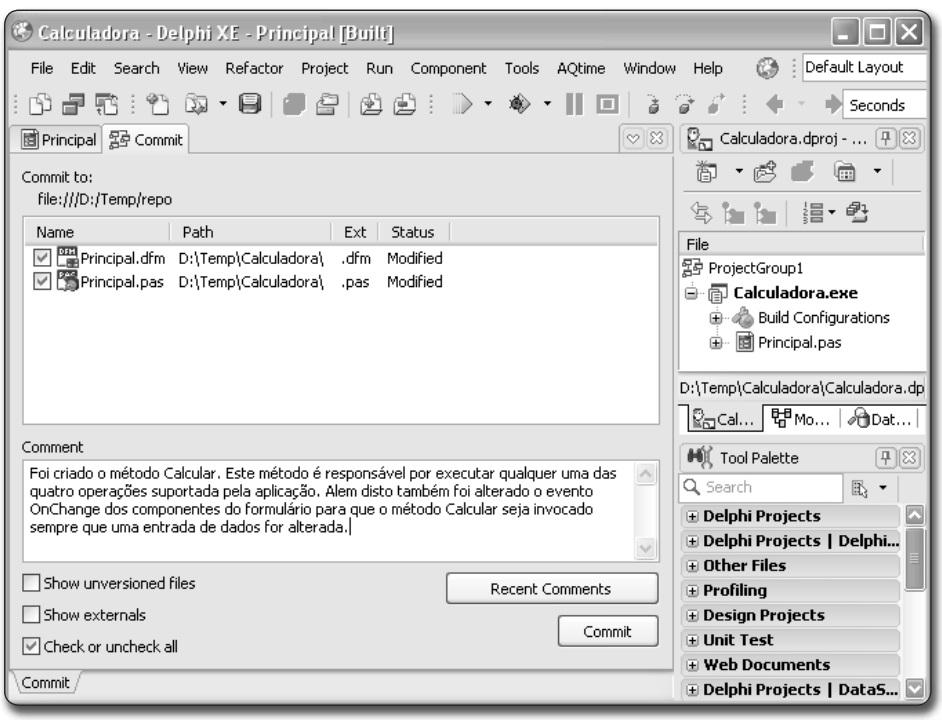


Figura 12. Submetendo as alterações

ao invés disto suas alterações serão mescladas com a última versão existente.

Para entender melhor, imaginemos que o arquivo Principal.pas do working copy está na versão 3 e no repositório ele está na versão 4, pois algum outro programador já submeteu alterações neste arquivo. Quando tentarmos submeter alterações feitas na versão 3 que possuímos, o SVN irá recusar pois o arquivo está desatualizado em relação ao arquivo do repositório.

Nesta situação devemos atualizar o arquivo Principal.pas para a versão 4. O Update é capaz de identificar as diferenças entre cada versão e juntá-las sem perder nada. Então, para submeter alterações do arquivo que está na versão 3, primeiro devemos atualizá-lo para verão 4. Ao fazer isto o SVN vai mesclar o arquivo do repositório que está na versão 4 com o arquivo local preservando suas alterações. Todo esse processo está ilustrado nas **Figuras 14 à 19** para ajudar a entender.

Na **Figura 14**, os três programadores atualizam a versão do arquivo que irão alterar, para ter certeza de que não estão alterando uma versão muito antiga. Esta atualização também é feita com o Update. A seguir – **Figura 15** – cada programador altera o que precisa alterar. O detalhe é que mesmo sem saber, os programadores João e Marcio estão alterando o mesmo arquivo simultaneamente, apesar de ser em linhas diferentes. Depois de concluir as alterações, Kelly e João fazem o Commit das alterações que fizeram, enquanto isto, Marcio continua alterando a linha 562 do arquivo C (**Figura 16**). Repare que agora Marcio está alterando um arquivo desatualizado, pois João já submeteu uma nova versão do arquivo C para o repositório.

Quando Marcio tenta executar o Commit o Subversion recusa, avisando que a versão do arquivo C alterada pelo Marcio está desatualizada (**Figura 17**). Para solucionar o problema, Marcio deve atualizar sua versão local (**Figura 18**). Ele faz isto executando o Update, que vai mesclar a alteração feita por João na linha 100 com o arquivo existente no working copy do Marcio (**Figura 19**). Após atualizar a versão do arquivo, Marcio está apto a submeter alterações feitas no arquivo C. Ao final disto tudo, o arquivo A alterado pela Kelly terá uma nova versão, e o arquivo C alterado por João e Marcio, terá duas novas versões.

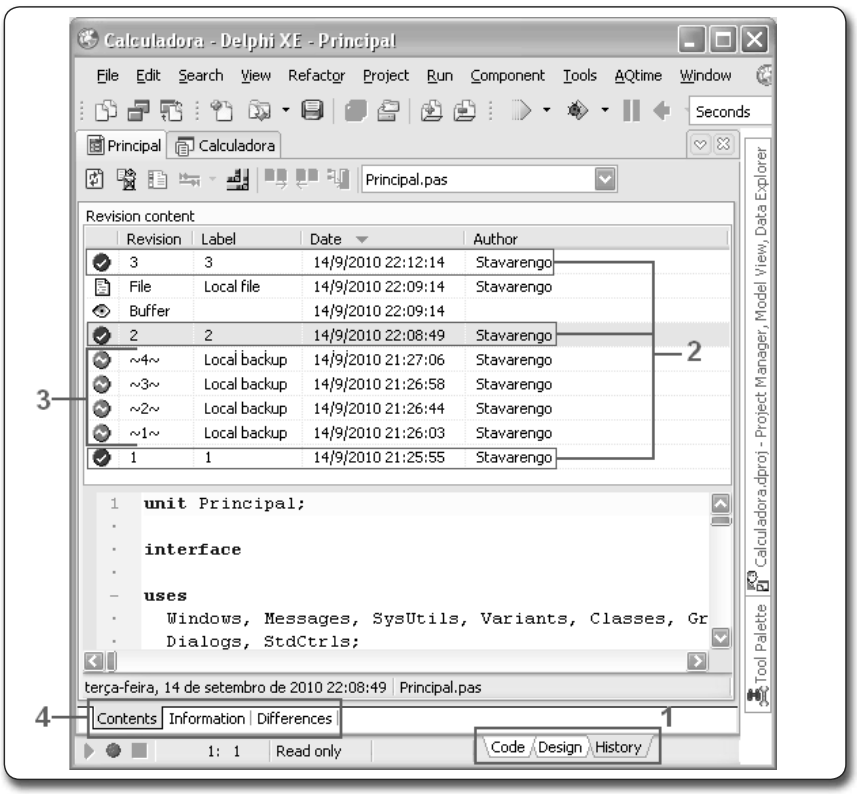


Figura 13. Histórico de versões

Nota do DevMan

Cada vez que um arquivo é salvo dentro do IDE do Delphi, um cópia do arquivo é criada dentro da pasta __history. Esta pasta é criada automaticamente pelo Delphi no local onde o projeto está salvo. Este recurso não é muito interessante quando o projeto já está sob o controle de um sistema controlador de versão. Para desativá-la, é preciso acessar Tools + Options do Delphi e dentro da opção Editor options desmarcar o item Create backup files.

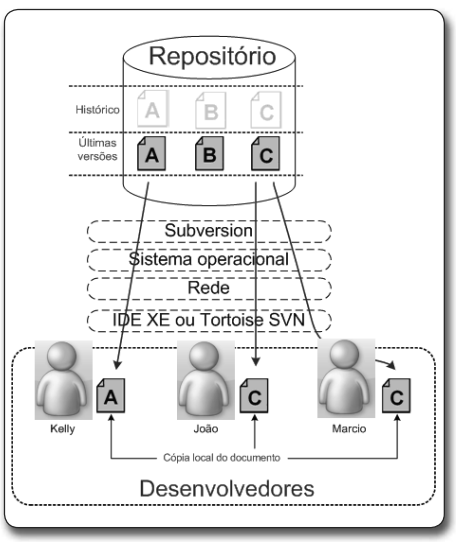


Figura 14. Os programadores executam update no seu working copy

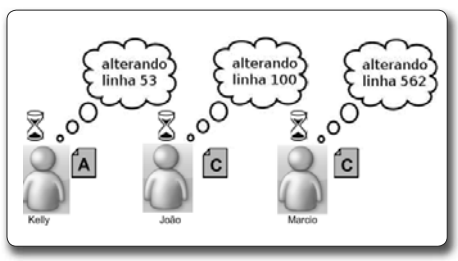


Figura 15. Cada programador altera o que precisa

Conflito: um arquivo, dois programadores

Uma vantagem do SVN é que mais de uma pessoa pode trabalhar no mesmo arquivo ao mesmo tempo já que cada

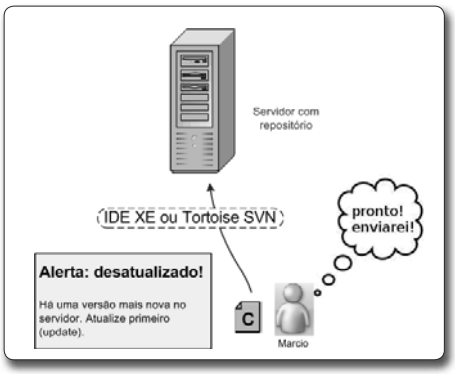


Figura 16. Submetendo as alterações (commit)

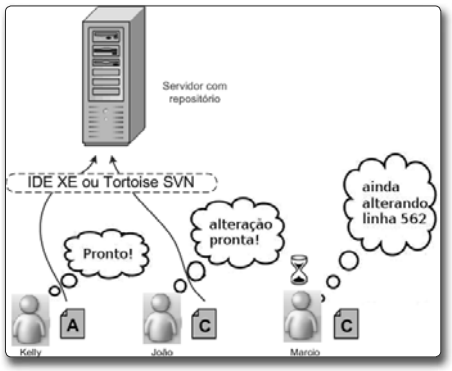


Figura 17. Conflito de edição

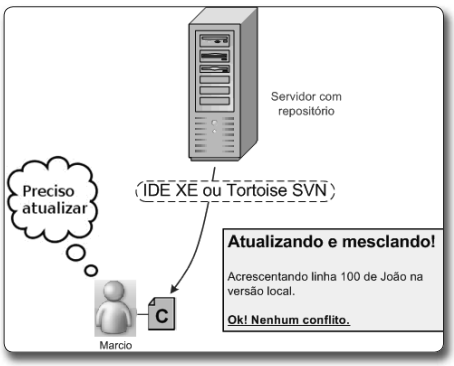


Figura 18. Outro update - Mesclando as alterações

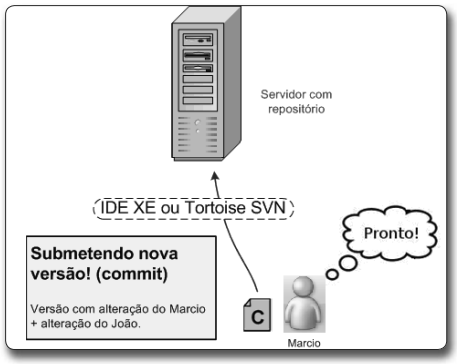


Figura 19. Submetendo nova versão (commit)

pessoa possui uma cópia dos arquivos em sua própria máquina. Quando isto acontecer ocorrerá um conflito de edição. O Subversion é um sistema controlador de versão que trabalha com mesclagem otimista. Esta filosofia de mesclagem acredita que os conflitos de edições serão tão raros que a mesclagem pode ser feita de forma automática. A primeira impressão é que esta filosofia não funciona, mas na prática é assim que os projetos vêm sendo gerenciados com o Subversion desde o início.

Esses conflitos podem ser separados em duas categorias: conflitos de arquivos e conflitos de linhas dos arquivos. Os conflitos de arquivos são comuns e acontecem quando dois programadores alteram o mesmo arquivo porém em linhas diferentes. Neste caso o Subversion será capaz de mesclar as alterações automaticamente. Quando dois programadores fazem alterações na mesma linha do mesmo arquivo, o Subversion não será capaz de fazer a mesclagem sozinho e você deverá dizer para o Subversion qual das duas edições será descartada.

Conclusão

Gerenciar o projeto usando O SVN é uma prática excelente. Em pouco tempo usufruímos das vantagens, como por exemplo, o histórico de vida do projeto, que é a vantagem mais evidente. Além disto ganhamos a capacidade trabalhar em equipe sem se preocupar com conflito de versões, pois o SVN mescla as alterações automaticamente.

Existe um livro chamado Version Control With Subversion, este é o livro oficial do Subversion e gratuito. Além disto já existe a versão em português para download (veja na sessão links). Aqui aprendemos como usar o Subversion usando o novo Delphi XE, mas se tiver interesse em aprender mais sobre como funciona o Subversion recomendo a leitura do livro.

Rafael Stavarengo
faelsta@gmail.com
http://www.devmedia.com.br/stavarengo
Programador-analista da Cheina Informática. Trabalha com Delphi, JME, JSE e PHP + ZendFramework. Formando em Tecnologia de Análise e Desenvolvimento de Sistemas pela UNIPAR.

Nota

As **Figuras 1, 2, 14, 15, 16, 17, 18** e **19**, foram baseadas em imagens encontradas no site de conteúdo livre Wikipédia: http://www.wikipedia.org.

Página oficial do TortoiseSVN
http://tortoisesvn.tigris.org/

Página oficial do Subversion
http://subversion.tigris.org/

Site oficial do projeto ACBr
http://acbr.sourceforge.net/drupal/

Site do livro oficial do SVN
http://svnbook.red-bean.com/

Site do oficial do livro SVN em português
http://code.google.com/p/svnbook-pt-br/

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Design Patterns – Parte 3

Vitor Luiz Rubio

Resumo DevMan

De que trata o artigo:

Será mostrado como usar os conhecimentos adquiridos nos artigos anteriores para criar um programa independente de banco de dados ou de tecnologia de acesso, tornando tudo isso configurável. Serão utilizados os padrões Singleton, Factory Method e Abstract Factory.

Para que serve:

Abstract Factory é um modo de desacoplar ainda mais os sistemas, por delegar a criação de famílias inteiras de objetos sem se especificar uma classe concreta e usando uma única interface. Esse padrão permite decidir "tardiamente" qual a classe do objeto que será instanciado, tornando possível que várias partes do sistema sejam intercambiáveis.

Em que situação o tema é útil:

Pode ser usado em qualquer situação onde há um conjunto de objetos que podem ser substituídos entre si para atender a um requisito. Serão aplicados esses conceitos na criação, configuração e conexão com bancos de dados. Porém não se aplica somente a isso. Num sistema de comunicações internas, por exemplo, pode-se decidir se uma mensagem é um SMS ou um e-mail em tempo de execução. Também é possível adiar a criação de gráficos após decidir pelo uso ou não de uma determinada API quando estiver criando games por exemplo.

Design Patterns – Parte 3:

Como no Delphi nós já temos a VCL e o sistema multiplataforma não é o foco desse artigo, esses conceitos serão aplicados tanto à conexão com o banco de dados como à escolha de que tipo de componente usar. Sistemas modernos podem ser configurados com qualquer banco de dados. O tipo de SGBD utilizado nunca deveria ser pré-requisito ou fator decisivo para a implementação de um projeto, já que dados são uma coisa, de propriedade do cliente, e o sistema é outra coisa. Os sistemas não só devem ser flexíveis a ponto de utilizar diversas bases de dados como devem fazê-lo instantaneamente, ou até simultaneamente, integrando sistemas distintos.

Muito se comenta sobre a orientação a objetos e sua importância em aplicações bem estruturadas, flexíveis, robustas e de qualidade. Embora seja sempre comentada, a orientação a objetos ainda não é algo frequentemente utilizado pela comunidade Delphi, porém, esta série de artigos vem abordar um dos principais temas relacionados à POO e padrões de projeto. O intuito de tudo isto é fazer com que o leitor adote a orientação a objetos como uma forma de desenvolvimento cotidiano, fazendo com que qualquer aplicação ou problema seja resolvido de uma forma mais ampla, que permita facilidade de manutenção e poupe menos tempo.

Como já sabemos, os padrões de projeto foram criados para revolucionar a forma de se desenvolver softwares, sendo adotados principalmente no segmento de orientação a objetos. Estudar padrões de projeto não só torna o seu desenvolvimento muito mais produtivo, como também faz com que você possua muito mais interesse em se aprofundar na orientação a objetos e desenvolver softwares que utilizem estas metodologias e conceitos.

Nos artigos anteriores foram explorados muito os padrões Factory Method e Abstract Factory. Lembrando que o Factory Method é um método que se responsabiliza pela criação (e possível configuração ou ligação) de um objeto ou de um grupo de objetos. Ainda comentando sobre a importância da POO em aplicações flexíveis, vimos como o padrão Abstract Factory pode ajudar na flexibilidade do projeto, por postergar a decisão de que famílias de objetos criam para o cliente dos objetos propriamente ditos.

Além de continuar a explorar o padrão Abstract Factory, estudaremos outros padrões como o Singleton (já discutido), Mediator e Adapter, implementando-os como soluções para alguns dos problemas que teremos adiante. Será vista uma forma simples de se implementar um Singleton, e que é uma alternativa elegante, segura e encapsulada para variáveis públicas globais.

Este artigo irá abordar a base para a criação de um sistema multibanco, utilização de métodos de fábrica e fábricas abstratas para criação de objetos de conexão com o banco de dados, configuração da SQL, criação de DataSets corretos de acordo com

o tipo, ligação ao Database e configuração dos Providers para que os dados possam ser acessados e modificados.

Note que todas estas operações serão realizadas através da orientação a objetos, utilizando padrões de projeto e com muito pouco código. Veremos como é simples adotar Design Patterns e como eles podem tornar a nossa vida muito mais simples, trazendo enorme flexibilidade e facilidade de expansão de um projeto.

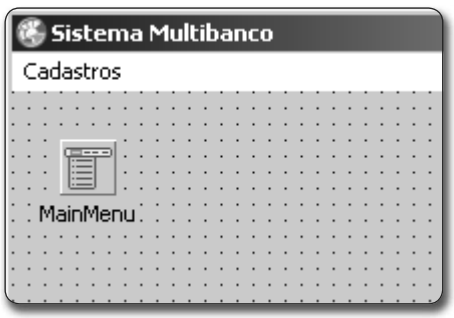


Figura 1. Formulário principal da aplicação

Nota do DevMan

O Mediator é um dos padrões de projeto para desenvolvimento de softwares orientados a objetos. Este permite que haja a "mediação" de um sistema por meio de uma interface. Esse padrão enquadra-se nos padrões comportamentais devido à maneira como ele pode alterar a execução do software. Para entender de uma forma clara, conforme um projeto cresce, suas classes crescem exponencialmente de forma natural, porém, as classes sempre possuem códigos e lógicas distribuídas entre si. Conforme o crescimento destas classes a comunicação entre as mesmas será cada vez mais complexa, havendo uma dependência muito maior entre elas e dificultando na manutenção do projeto. O padrão Mediator vem como sugestão para resolução deste problema, encapsulando a comunicação entre estes objetos e reduzindo a dependência entre si.

Nota do DevMan

Um Wrapper nada mais é que um outro nome dado a um padrão de projeto também conhecido como Adapter. Mais o que seria Adapter? Logo quando está se iniciando no mundo da orientação a objetos, sempre somos levados a pensar de uma forma comparativa ao mundo real, logo, transpondo isto para o mundo real, um adaptador nada mais seria que algo que servisse de "intermediador" parar tornar possível a compatibilidade entre objetos até então incompatíveis. Tratando-se de orientação a objetos em si, imagine que você está dando manutenção a um software já existente e deve criar uma nova classe que se comunique com classes existentes, porém, as interfaces de ambas as classes são diferentes, o que impossibilita essa comunicação "direta". O padrão Adapter resolve este problema, pois, neste caso seria o "intermediador" encarregado de compreender e tornar possível a comunicação entre elas. Ele nada mais faz do que converter uma interface de uma classe para outra interface que o cliente espera encontrar, fazendo com que interfaces incompatíveis trabalhem juntas.

Apresentação do exemplo

O exemplo do artigo (**Figura 1**) mostra uma aplicação multibanco e multicomponente, ou seja, é possível trocar não só o banco de dados, mas também os componentes de acesso aos mesmos. Isso permitirá usar a melhor tecnologia para cada banco e ainda poder repassar o custo da aquisição de algum novo componente de acesso aos dados exótico para o cliente. Tudo o que deve ser feito é uma nova classe para acesso a dados com o novo componente.

Este exemplo não tem por intenção ser um framework de persistência, até porque seria extenso demais. O objetivo é apenas demonstrar como é possível usar Factory Methods e outros Design Patterns para construir aplicações OO extensíveis e adaptáveis. Muitas das "Best Practices" (melhores práticas) conhecidas não foram usadas, por questões de objetividade.

A unit chamada uIDataSet define a interface IDataSet. Esta não terá a interface básica de um DataSet, mas as classes que a implementarem conterão descendentes de vários tipos de TDataSet, dependendo do banco de dados. A principal parte dessa interface é manter um método que aceite como parâmetro um TDataSetProvider.

O DataSetProvider recebe um DataSet na sua propriedade correspondente. Supondo que exista uma instância qualquer chamada umIDataSet, não pode-se atribuir diretamente DataSetProvider1.Dataset := umIDataSet porque não sabemos o que é um IDataSet, nem se é realmente um TDataSet. Por isso essa configuração deve ser feita internamente na classe que implementar a interface. Ou seja, é IDataSet que vai setar a propriedade DataSet do Provider e não o contrário. Nesses casos, para se evitar acoplamentos e Type Casts dos dois lados, seria bom usar um Mediator.

Nota do DevMan

O padrão Singleton é uma convenção para garantir que um e apenas um objeto seja instanciado para cada classe. O padrão Singleton também nos dá um ponto de acesso global, assim como uma variável global, mas sem as desvantagens. Se você atribuir um objeto a uma variável global, esse objeto poderá ser criado quando seu aplicativo for iniciado. E se esse objeto tiver muitos recursos e seu aplicativo nunca acabar de usá-lo? O padrão Singleton é uma das soluções para este problema, pois garante que os nossos objetos sejam criados somente quando necessários. É importante dominar bem os conceitos de variáveis e métodos de classe estática, assim como os modificadores de acesso.

A propriedade SQL do DataSet também será configurada "por dentro". A interface recebe com o método SetSQL o comando SQL que irá configurar internamente o DataSet do objeto que a implementar (TSQLDataSet ou TADODataSet, no exemplo).

São usados três padrões de projeto no exemplo: Factory Method, Abstract Factory e o Mediator. Os dois primeiros já foram exaustivamente comentados nesta série. Uma das aplicações para o Mediator que podemos dar é intermediar a conexão de dois objetos de classes distintas com um contrato em comum para que eles tenham baixo acoplamento entre si.

O projeto

Os objetos que implementam IDataSet (**Listagem 1**) recebem três parâmetros: um comando SQL, um objeto de conexão a banco de dados e um TDataSetProvider. O objetivo é claro: um ClientDataSet é ligado ao DataSetProvider, então todas as operações CRUD serão feitas através do ClientDataSet. O papel do IDataset será configurar o banco de dados e o Dataset por trás do DataSetProvider.

A classe TNossoADODataset (**Listagem 2**) é descendente de TInterfacedObject e implementa IDataSet. Foi usada uma interface comum e não um TObject ou TDataSet, pois como esse DataSet vai ser usado fora da fábrica (nos Forms), seria necessário saber seu tipo e fazer Type Cast apenas para setar suas propriedades SQL, CommandText, DataSet e Provider. Isso criaria acoplamento com as units da tecnologia (componente de acesso) em todos os Forms ou Data Modules, e a maior dificuldade em se trocar de banco de dados seria essa: trocar de tecnologia em todos os Forms. Usando interfaces tem-se a vantagem da contagem automática de referência, que libera o objeto da memória automaticamente.

Por exemplo, em alguns componentes de acesso a dados, a propriedade onde se informa a instrução SQL é chamada de SQL, em outras tecnologias poderia ser CommandText e até mesmo SQLCommand. O que está sendo criado é um "Wrapper" para toda e qualquer tecnologia de acesso a dados que siga uma interface comum.

Esta será herdada de TInterfacedObject e não de TADODataSet, pois este último é descendente de TComponent. TComponent implementa a interface básica IInterface e os contadores de referências de uma forma diferente, fazendo com que a mesma não seja liberada da memória e causando Memory Leaks. Ou pior, sendo previamente liberada causando Access Violations na hora de usar. Mas o motivo principal é criar total independência entre tecnologias do Delphi ou de terceiros.

A classe da **Listagem 2** possui um constructor, um destructor e os três métodos obrigatórios para que ela implemente IDataset. Também possui um campo privado FDataSet que é uma instância de um TADODataSet. Este é criado no constructor e destruído no destructor.

O método SetBancoDados pressupõe que o objeto bd seja um objeto TADOConnection válido e instanciado. Sendo nulo, será disparada uma exception.

O método SetSql muda o CommandText do FDataset e o método ConfiguraProvider coloca o FDataset ligado à propriedade DataSet do seu Provider. Ou seja, formulários, Data Modules e classes de negócio não precisam ter units e componentes relativos ao TADOConnection ou TADODataSet.

Usando a dupla ClientDataSet e DataSetProvider torna-se desnecessário se preocupar com certos detalhes de baixo nível sobre como os comandos CRUD são criados.

Após a implementação de uma classe que encapsula a conexão com ADO, temos uma equivalente para DBExpress (**Listagem 3**). Já que ambas tecnologias já são multibanco, basta apenas realizar as configurações mínimas para cada uma.

Na classe da **Listagem 3** é utilizado o método SetSql para mudar o CommandText e ConfiguraProvider para setar seu DataSet. É bom observar que em ambas as classes o método SetBancoDados recebe o parâmetro db do tipo TObject. TObject é um tipo muito genérico, porém, graças ao Abstract Factory e a um simples Type Cast pode-se garantir a criação segura do objeto. O cenário ideal ainda não foi criado, pois as classes ainda devem reconhecer suas respectivas conexões. Seria possível resolver isto com outra interface ou mesmo através do padrão Mediator.

O ideal é criar uma Abstract Factory, que sirva de base para todas as fábricas concretas, como mostra a **Listagem 4**. Essa fábrica abstrata já tem todas as propriedades que um banco precisa para se conectar, para que as outras fábricas herdem dela.

No caso as propriedades são:

- Tecnologia: define o tipo de componente que será usado para acessar o banco de dados;
- TipoBancoDados: o tipo do SGBD: Firebird, SQLServer, MySQL;
- Servidor: o Host ou IP onde está o banco de dados;
- NomeBanco: nome do banco de dados para bancos como SQLServer ou MySQL

Listagem 1. Interface IDataset

```
unit uIDataset;

interface

uses
  provider;

type
  IDataSet = interface
    ['{B3A8D9C9-2466-43F7-B61D-3554
    EDB06945}']
    procedure SetSql(sql: string);
    procedure SetBancoDados(bd: TObject);
    procedure ConfiguraProvider
    (prov: TDataSetProvider);
  end;

implementation

end.
```

Nota do DevMan

Outros patterns comumente utilizados em Delphi

Command

O padrão Command faz parte dos padrões de projeto. Seu objetivo é encapsular qualquer ação que venha a ser chamada em um momento futuro. Essa ação torna-se um objeto que pode ser reutilizado, contendo em si a complexidade da chamada da ação original. Ao aplicar esse padrão conseguimos uma flexibilidade maior no sistema, já que para adicionar novas ações (comandos) não há necessidade de se alterar o design do projeto, apenas novos objetos são criados e associados à interatividade do sistema. Apesar dessa vantagem existe uma desvantagem no seu uso. Como seu uso implica na execução de objetos que sua única finalidade é executar uma ação, acabaremos por ter um código que se assemelha muito ao código procedural, orientado aos comandos.

Builder

O Builder é um padrão de projeto que permite a separação da criação de um objeto da sua representação, de forma que este permita a construção de diferentes representações. Este padrão é muitas vezes comparado com ao Abstract Factory pois ambos podem ser utilizados para a construção de objetos complexos. A grande diferença entre ambos é que o Builder constrói apenas objetos complexos enquanto o Abstract Factory constrói famílias de objetos, simples ou complexos, de uma só vez.

Flyweight

O objetivo do padrão Flyweight é minimizar o uso de memória no armazenamento de vários objetos através do compartilhamento das informações em comum que essas instâncias possuem. A tradução do nome desse padrão é "peso-mosca", categoria do boxe onde os lutadores têm entre 49 e 51 quilos. É essa a ideia da aplicação do pattern: fazer com que tenhamos aplicações leves.

Quando trabalhamos com orientação a objetos, utilizamos classes para representar cada parte das aplicações. Por exemplo, um sistema de contabilidade terá objetos que representam duplicatas, notas fiscais, itens da nota etc. Um editor de textos provavelmente terá objetos que representam parágrafos, frases, caracteres, imagens etc. Com essa abordagem temos vantagens da programação orientada a objetos, como herança, polimorfismo e encapsulamento, no entanto podemos introduzir um gasto na manutenção de toda essa estrutura de objetos. É nesse problema que o Flyweight atua.

Um conceito chave para a utilização desse padrão é a distinção entre dados intrínsecos e dados extrínsecos. Dados intrínsecos são as informações comuns entre as várias instâncias e que por essa característica podem ser compartilhadas. Já os dados extrínsecos são as informações específicas de cada objeto e que consequentemente não podem ser compartilhadas. Os dados intrínsecos são mantidos nos objetos que implementam o Flyweight, e os extrínsecos são mantidos pelo cliente.

Este padrão é especialmente indicado para os casos de aplicações com um grande número de objetos com muitas informações que devem ser armazenados em memória. Do outro lado, ele não deve ser utilizado em situações em que precisamos fazer comparações entre os objetos, já que como eles serão compartilhados, esse tipo de ação se torna inviável (as comparações sempre retornarão que os objetos são os mesmos!).

Strategy

O padrão Strategy consiste em se possuir vários algoritmos, ou estratégias, encapsulados em uma classe chamada Context. O programa cliente pode selecionar um desses diferentes algoritmos, de acordo com a situação. Em alguns casos, a classe Context pode até decidir qual é a melhor no momento. O interessante é que essa escolha é feita em tempo de execução.

Com esse padrão, podemos mais facilmente adicionar novos algoritmos, alterar os já existentes ou estender suas funcionalidades.

e caminho do banco para Interbase/Firebird;
• Login: usuário do banco;
• Senha: password do banco.

O segredo da fábrica TAbstractFactory são os métodos CriaBancoDados, que cria o objeto de conexão com o banco, e o CriaDataSet, que cria uma instância de um objeto qualquer que implemente IDataSet.

Servidor, NomeBanco, Login e Senha são propriedades do objeto de banco de dados e não da fábrica abstrata. No entanto para isso é preciso outra família de fábricas e outro Mediator, além de fazer algum buffer do arquivo INI (mostrado no artigo anterior) para não acessá-lo duas vezes.

A lógica toda foi centralizada na fábrica para simplificar, porém, quanto mais descentralizar, mais flexível ficará caso seja necessária uma alteração futura.

A fábrica de bancos de dados tem algumas responsabilidades adicionais,

Listagem 2. IDataset com ADO

```
unit uNossoADODataset;

interface

uses
  ADODB, uIDataset, provider;

type
 TNossoAdoDataset = class(TInterfacedObject, IDataset)
 private
   FDataset:  TADODataSet;
 public
   constructor Create; virtual;
   destructor Destroy; override;
   procedure SetSql(sql: string);
   procedure SetBancoDados(bd: TObject);
   procedure ConfiguraProvider(prov: TDataSetProvider);
 end;

implementation

uses Classes, SysUtils;

{ TNossoAdoDataset }

procedure TNossoAdoDataset.ConfiguraProvider
 (prov: TDataSetProvider);
begin
  prov.DataSet := FDataset;
end;

constructor TNossoAdoDataset.Create;
begin
  inherited;
  FDataset := TADODataSet.Create(nil);
end;

destructor TNossoAdoDataset.Destroy;
begin
  FDataset.Free;
  inherited;
end;

procedure TNossoAdoDataset.SetBancoDados(bd: TObject);
begin
  if bd <> nil then
  begin
     FDataset.Connection := (bd as TADOConnection);
  end
  else
    raise Exception.Create('Objeto banco de dados nulo!');
end;

procedure TNossoAdoDataset.SetSql(sql: string);
begin
  FDataset.CommandText := sql;
end;

end.
```

Listagem 3. IDataset com DBExpress

```
unit uNossoDBXDataset;

interface

uses
  SqlExpr, uIDataset, provider;

type
 TNossoDbxDataset = class(TInterfacedObject, IDataset)
 private
   FDataset: TSQLDataSet;
 public
   constructor Create; virtual;
   destructor Destroy; override;
   procedure SetSql(sql: string);
   procedure SetBancoDados(bd: TObject);
   procedure ConfiguraProvider(prov: TDataSetProvider);
 end;

implementation

uses
  sysutils;

{ TNossoDbxDataset }

procedure TNossoDbxDataset.ConfiguraProvider(prov: TDataSetProvider);
begin
  prov.DataSet := FDataset;
end;

constructor TNossoDbxDataset.Create;
begin
  inherited;
  FDataset := TSQLDataSet.Create(nil);
end;

destructor TNossoDbxDataset.Destroy;
begin
  FDataset.Free;
  inherited;
end;

procedure TNossoDbxDataset.SetBancoDados(bd: TObject);
begin
  if bd <> nil then
  begin
    FDataset.SQLConnection := (bd as TSQLConnection);
  end
  else
    raise Exception.Create('Objeto banco de dados nulo!');
end;

procedure TNossoDbxDataset.SetSql(sql: string);
begin
  FDataset.CommandText := sql;
end;

end.
```

mas que também podem ser isoladas. As responsabilidades são as seguintes:

1) Retornar um objeto de conexão ao banco de dados já configurado, podendo já estar conectado ou não;

2) Retornar DataSets compatíveis com o objeto de banco de dados instanciado já configurados com o objeto de banco de dados instanciado previamente;

3) Fornecer uma maneira de o objeto de banco de dados ser um Singleton, sem usar variáveis públicas globais, e que este seja liberado da memória quando a sua fábrica for liberada.

Para criar dois objetos de bancos de dados diferentes, deve-se criar duas fábricas concretas. Cada fábrica retornará DataSets conectados ao seu objeto de banco. Pode-se usar bancos de dados diferentes.

A implementação do exemplo mostra o acesso ao SQLServer com ADO, e ao Firebird com DBExpress (note que a melhor tecnologia é empregada em cada caso). Pode-se também fazer cada um dos dois acessar os dois tipos de banco de dados. Além disso, é possível implementar UIB, Zeos, IBO, ou o que quiser.

O mais importante é que os Forms e Data Modules não tenham conhecimento das units ou componentes que acessam os bancos de dados, permitindo uma migração muito fácil.

Listagem 4. Classe TAbstractFactory para banco de dados

```
unit uAbstractFactory;

interface

uses
  ulDataset;

type
  TAbstractFactory = class abstract
  private
    FServidor: string;
    FSenha: string;
    FLogin: string;
    FTecnologia: string;
    FNomeBanco: string;
    FTipoBancoDados: string;
    procedure SetLogin(const Value: string);
    procedure SetNomeBanco(const Value: string);
    procedure SetSenha(const Value: string);
    procedure SetServidor(const Value: string);
    procedure SetTecnologia(const Value: string);
    procedure SetTipoBancoDados(const Value: string);
    function GetServidor: string;
    function GetNomeBanco: string;
    function GetLogin: string;
    function GetSenha: string;
    function GetTecnologia: string;
    function GetTipoBancoDados: string;
  public
    function CriaBancoDados: TObject; virtual; abstract;
    function CriaDataset: IDataset; virtual; abstract;
    property Servidor: string
      read GetServidor write SetServidor;
    property NomeBanco: string
      read GetNomeBanco write SetNomeBanco;
    property Login: string
      read GetLogin write SetLogin;
    property Senha: string
      read GetSenha write SetSenha;
    property Tecnologia: string
      read GetTecnologia write SetTecnologia;
    property TipoBancoDados: string
      read GetTipoBancoDados write SetTipoBancoDados;
  end;

implementation

{ TAbstractFactory }

function TAbstractFactory.GetLogin: string;
begin
  Result := FLogin;
end;

function TAbstractFactory.GetNomeBanco: string;
begin
  Result := FNomeBanco;
end;

function TAbstractFactory.GetSenha: string;
begin
  Result := FSenha;
end;

function TAbstractFactory.GetServidor: string;
begin
  Result := FServidor;
end;

function TAbstractFactory.GetTecnologia: string;
begin
  Result := FTecnologia;
end;

function TAbstractFactory.GetTipoBancoDados: string;
begin
  Result := FTipoBancoDados;
end;

procedure TAbstractFactory.SetLogin(const Value: string);
begin
  FLogin := Value;
end;

procedure TAbstractFactory.SetNomeBanco(const Value: string);
begin
  FNomeBanco := Value;
end;

procedure TAbstractFactory.SetSenha(const Value: string);
begin
  FSenha := Value;
end;

procedure TAbstractFactory.SetServidor(const Value: string);
begin
  FServidor := Value;
end;

procedure TAbstractFactory.SetTecnologia(const Value: string);
begin
  FTecnologia := Value;
end;

procedure TAbstractFactory.SetTipoBancoDados(const Value: string);
begin
  FTipoBancoDados := Value;
end;

end.
```

É importante usar comandos SQL em comum para todos os bancos. Se for usar alguma coisa muito particular de algum banco, lembre-se que a Factory desse banco deve ser capaz de conhecer, criar e usar o recurso que for necessário.

Duas classes (TADOFactory e TDBXFactory) usam os componentes da família ADO (dbGo) e DBExpress respectivamente. A classe da **Listagem 5**, para ADO, possui a implementação do método CriaBancoDados. Este método cria um TADOConnection apenas se o campo privado FBancoDados for nil. Caso contrário retorna o objeto já criado. Isso é importante para que a conexão com o banco seja um Singleton. Usando um componente TADOConnection em um Data Module ou formulário seria necessário apontar todos os DataSets espalhados para ele. Se fossem criados mais de um formulário ou Data Module desses, ou mais de um componente de conexão, haveria duas conexões.

Foi desenvolvida uma forma de Singleton porque uma conexão com banco de dados pode ser um recurso escasso. Além disso, a licença de uso do banco de dados pode ser cobrada por conexão. É importante saber que este modelo não atende quem necessita de mais de uma conexão simultânea na mesma instância de aplicativo, ou de conexão a mais de um banco de dados ao mesmo tempo.

Ainda é responsabilidade do método CriaBancoDados montar a string de conexão com o banco baseado nos outros atributos da classe preenchidos através de consulta a um arquivo INI. Nesse ponto um dos parâmetros que compõe a connection string foi colocado "hardcoded" no código fonte para fins de simplicidade, porém esses parâmetros podem ser armazenados externamente.

Para um dos parâmetros que foi configurado no Provider para SQL Server há três possibilidades:

- Provider=SQLOLEDB.1 - Microsoft OleDb Provider for SQL Server
- Provider=SQLNCLI.1 - SQL Native Client
- Provider=SQLNCLI10.1 - SQL Native Client 10.0

Listagem 5. Classe TADOFactory

```
unit uADOFactory;

interface

uses
  uAbstractFactory, uldataSet, ADODB;

type
  TADOFactory = class(TAbstractFactory)
  private
    FBancoDados: TADOConnection;
  public
    function CriaBancoDados: TObject; override;
    function CriaDataset: IDataset; override;
    destructor Destroy; override;
  end;

implementation

uses
  uNossoADODataset, sysutils;

{ TADOFactory }

function TADOFactory.CriaBancoDados: TObject;
begin
  {
    para simular um singleton, cria somente se FBancoDados = nil,
    senão reaproveita o objeto criado
  }
  //exemplo de connection string:
  //Provider=SQLNCLI.1;Password=123456;Persist Security Info
    =True;User ID=sa;Initial Catalog=AdventureWorks;Data Source
    =localhost;Use Procedure for Prepare=1;Auto Translate
    =True;Packet Size=4096;Workstation ID=VITOR;Use Encryption for Data
    =False;Tag with column collation when possible
    =False;MARS Connection=False;DataTypeCompatibility
    =0;Trust Server Certificate=False
  //possíveis providers para SQLServer:
  //Provider=SQLOLEDB.1 - microsoft OleDb Privider for SQL Server
  //Provider=SQLNCLI.1 - SQL Native Client
  //Provider=SQLNCLI10.1 - SQL Native Client 10.0
  if FBancoDados = nil then
  begin
    if TipoBancoDados = 'Firebird' then
    begin
      FBancoDados := TADOConnection.Create(nil);
      FBancoDados.LoginPrompt := false;
      FBancoDados.ConnectionString := 'DSN=Firebird;Driver
      =Firebird/InterBase(r) driver;' +
      'Dbname='+Servidor+':'+ NomeBanco +';' +
      'CHARSET=NONE;'+
      'PWD='+Senha+';'+
      'UID='+Login+';'+
      'Client=C:\WINDOWS\system32\FBCLIENT.DLL;';
    end
    else if TipoBancoDados = 'SQLServer' then
    begin
      FBancoDados := TADOConnection.Create(nil);
      FBancoDados.LoginPrompt := false;
      FBancoDados.ConnectionString := 'Provider=SQLNCLI10.1;'+
                                      'Password='+Senha+';'+
                                      'User ID='+Login+';'+
                                      'Initial Catalog='+NomeBanco+';'+
                                      'Data Source='+Servidor+';';
    end
    else
      raise Exception.Create('Tipo de banco de dados não configurado
      para esta tecnologia');
  end;
  Result := FBancoDados;
end;

function TADOFactory.CriaDataset: IDataset;
begin
  Result := TNossoAdoDataset.Create;
  Result.SetBancoDados(CriaBancoDados as TADOConnection);
end;

destructor TADOFactory.Destroy;
begin
  FBancoDados.Free;
  inherited;
end;

end.
```

O mesmo vale para Providers para usar com o Firebird. Na seção de Links encontram-se tanto drivers ODBC como drivers OLEDB para conectar o TADOConnection ao Firebird. Já o DBExpress já vem com drivers nativos para SQL e Firebird a partir da versão Architect do Delphi 2010 ou XE.

No método CriaBancoDados a montagem da string de conexão para bancos SQL Server e para Firebird via ODBC é ligeiramente diferente, embora o princípio seja o mesmo. Em ambos os casos são usados alguns parâmetros codificados rigidamente, pois tem pouca relevância nesse exemplo, mas a connection string inteira foi construída concatenando os atributos lidos do arquivo INI. O método CriaDataSet cria um objeto do tipo TNossoAdoDataSet e o associa com o banco de dados.

A Concrete Factory para ADO cria dois tipos de objetos da mesma família: o objeto de conexão com o banco TADOConnection e o DataSet compatível.

A classe concreta para DBExpress é análoga e pode ser vista na **Listagem 6**.

Particularmente no uso de DBExpress é necessário que a unit uDBXFactory conheça algumas units no uses:

- Dbxmssql – para se trabalhar com SQL;
- Dbxinterbase – Se desejar trabalhar com Interbase;
- Dbxfirebird – para se trabalhar com Firebird;

Existem outras units desses tipos para outros bancos de dados. Outra coisa necessária às conexões com DBX são as DLLs de drivers DBX e as VendorLibs. Trabalhando com SQL 2008 são necessárias as DLLs dbxmss.dll e sqlncli10.dll no diretório do aplicativo ou na pasta System32 do Windows. Trabalhando com Firebird são necessárias as DLLs dbxfb.dll e fbclient.dll, e assim por diante.

Uma diferença considerável na classe da **Listagem 6** é que certos parâmetros da tecnologia DBExpress precisam ser configurados. E alguns desses parâmetros podem ser diferentes dependendo do banco de dados a ser acessado.

Em alguns desses parâmetros, como Vendorlib, optou-se por ser colocado "hardcoded" direto no código fonte, mas o ideal é que isso seja configurável pelo arquivo .INI e que o método fábrica já retorne o objeto "montado". Vendorlib é a DLL responsável pela conexão do DBExpress ao banco, é o driver cliente fornecido pelo fabricante do SGBD.

Uma unit simula um Data Module e cria apenas uma instância de uma fábrica concreta, através de métodos de classe, simulando um Singleton. A **Listagem 7** ilustra a criação de uma unit responsável pela criação da fábrica concreta correta de acordo com os parâmetros configurados no arquivo INI.

Os métodos CriaBancoDados e CriaDataSet são apenas atalhos para os métodos de mesmo nome do objeto privado FFactory. O método InicializaBD é o responsável por criar o objeto FFactory e popula seus atributos com os dados lidos do arquivo INI. Primeiro é checado o arquivo INI na seção [config] para saber qual tecnologia será usada. Em seguida, este cria a fábrica concreta do tipo desejado para checar o tipo do banco na seção [<<Tecnologia que encontrou>>]. Por último são verificados os parâmetros de conexão: login, senha, host e nome do banco que estão na seção [<<Tipo de banco de dados encontrado>>].

FinalizaBd destrói FFactory. A propriedade de classe BdFactory expõe a instância FFactory. Todos os atributos dessa classe, privados ou não, são estáticos (de classe). Isso significa que essa classe não precisa ser instanciada. Classes desse tipo não proveem um tipo de dado ou entidade, mas sim um serviço.

As seções initialization e finalization da **Listagem 7** criam e destroem respectivamente a fábrica do banco de dados. Todo esse processo assegura que tanto o banco de dados como a fábrica de bancos de dados sejam Singletons. Garante também que a fábrica estará acessível a qualquer módulo do sistema depois que o programa iniciar.

Nota

Os bancos de dados nos quais o TADOConnection poderá se conectar vão depender dos drivers OLEDB e ODBC instalados. No exemplo são utilizadas conexões com Firebird e MS SQL Server 2008 Express. A única particularidade relativa aos drivers de conexão é referente ao ADO que utiliza driver ODBC da IBPhoenix. Existe também um driver OLEDB para Firebird, gratuito, da IbProvider. Esses SGBDs e drivers podem ser encontrados na seção Links ao final do artigo. É interessante lembrar que todos os parâmetros podem ser mudados pelo INI, bastando que o computador onde o sistema for instalado tenha o driver ODBC da IBPhoenix e a fbclient.dll na pasta System32 do Windows.

As **Listagens 8** e **9** mostram os comandos de criação do banco de dados, com os scripts tanto para Firebird como para SQL Server. A diferença entre as duas listagens é que no Firebird existe um Trigger e um Generator para fazer com que a chave primária seja "auto incremento".

A instrução SQL normalmente usada para acessar a tabela é:

```
select Nome, Telefone from contatos
```

Um detalhe interessante é que a chave primária não é incluída no SQL padrão. Isso facilita na hora dos comandos Updates, já que a chave é gerada automaticamente e não pode ser alterada pelo usuário, mas gera também um problema, pois não pode ficar em branco no ClientDataSet. Este problema é facilmente contornado configurando os ProviderFlags.

A **Listagem 10** mostra a configuração do arquivo INI. O arquivo completo com todas as suas possibilidades será exibido, mas todas essas configurações não são usadas em uma única conexão. Os parâmetros precedidos de ponto-e-vírgula ";" estão "comentados" e portanto desativados.

O INI é uma espécie de arquivo "recursivo" ou "encadeado". Caracteriza-se assim porque o valor da chave "Tecnologia" da seção "Config" pode ser DBX ou ADO (ou qualquer outra). Esta string é a chave para a leitura da próxima seção: DBX ou ADO que por sua vez contém

a chave TipoBancoDados que pode ter os valores SQLServer ou Firebird. Esses dois valores por sua vez definem outras sessões para se encontrar os parâmetros do banco de dados em questão.

O formulário da **Figura 2 / Listagem 11** é um cadastro que faz uso de DbFactory e IDataset. O OnCreate instancia o objeto FDataset (definido na sessão private) do tipo IDataSet com o método de fábrica TBdFactory.CriaDataSet. Em seguida a instrução SQL e o Provider são configurados. Os dados podem ser visualizados no DBGrid e o DBNavigator pode ser usado para as operações CRUD. O botão "Apply Updates" executa fisicamente as operações que estiverem na propriedade delta do ClientDataSet.

Existem quatros possibilidades de configuração no exemplo: DBExpress com Firebird, DBExpress com SQL Server, ADO com Firebird via ODBC e ADO com SQL Server. Isso possibilita quatro diferentes tipos de configura-

Listagem 6. Classe TDBXFactory

```
unit uDBXFactory;

interface

uses
 uAbstractFactory,
 uIdataSet,
 SQlExpr,
 dbxmssql,    //unit necessária para fuincionamento com SQL
 dbxinterbase, //unit necessária para funcionamento com interbase
 dbxfirebird;  //unit necessária para fuincionamento com firebird

type
 TDBXFactory = class(TAbstractFactory)
 private
  FBancoDados: TSQLConnection;
 public
  function CriaBancoDados: TObject; override;
  function CriaDataset:    IDataset; override;
  destructor Destroy; override;
 end;

implementation

uses
 uNossoDBXDataset, sysutils;

{ TDBXFactory }

function TDBXFactory.CriaBancoDados: TObject;
begin
 {
    para simular um singleton, cria somente se FBancoDados = nil,
    senão reaproveita o objeto criado
 }
 if FBancoDados = nil then
 begin
  FBancoDados := TSQLConnection.Create(nil);
  FBancoDados.LoginPrompt := false;
  if (TipoBancoDados = 'Firebird') then
  begin
   //aqui vamos setar as propriedades do driver dbExpress, mas você pode
   //tornar mais flexível usando o arquivo ini
   //nome de driver
    FBancoDados.DriverName := 'Firebird';
   //nome da dll do driver:
    FBancoDados.LibraryName := 'dbxfb.dll';
   //vendorlib:
    FBancoDados.VendorLib := 'fbclient.dll';
    FBancoDados.GetDriverFunc := 'getSQLDriverINTERBASE';
    FBancoDados.KeepConnection := false;
   //configuração dos parâmetros
    FBancoDados.Params.Values['DriverName'] := 'Firebird';
    FBancoDados.Params.Values['Database'] := Servidor+':'+NomeBanco;
    FBancoDados.Params.Values['Password'] := Senha;
    FBancoDados.Params.Values['User_name'] := Login;
   //não esqueça de configurar role name, charset e outras coisas que talvez
     seu sistema use
    FBancoDados.Params.Values['BlobSize']:='-1';
    FBancoDados.Params.Values['CommitRetain']:='False';
    FBancoDados.Params.Values['SQLDialect']:='3';
   FBancoDados.Params.Values['Interbase TransIsolation']:='ReadCommited';
    FBancoDados.Params.Values['WaitOnLocks']:='True';
  end
  else
  if (TipoBancoDados = 'SQLServer') then
  begin
   FBancoDados.DriverName := 'MSSQL';
   FBancoDados.LibraryName := 'dbxmss.dll';
   FBancoDados.VendorLib := 'sqlncli10.dll';
   FBancoDados.GetDriverFunc := 'getSQLDriverMSSQL';
   FBancoDados.KeepConnection := false;
   //configuração dos parâmetros
    FBancoDados.Params.Values['DriverName']:='MSSQL';
    FBancoDados.Params.Values['Password'] := Senha;
    FBancoDados.Params.Values['User_name'] := Login;
    FBancoDados.Params.Values['Database'] := NomeBanco;
    FBancoDados.Params.Values['HostName'] := Servidor;
   //não esqueça de configurar role name, charset e outras coisas que talvez
     seu sistema use
    FBancoDados.Params.Values['BlobSize']:='-1';
   FBancoDados.Params.Values['MSSQL TransIsolation']:='ReadCommited';
    FBancoDados.Params.Values['SchemaOverride']:='sa.dbo';
    FBancoDados.Params.Values['OS Authentication']:='False';
    FBancoDados.Params.Values['Prepare SQL']:='False';
  end
  else
   raise Exception.Create('Tipo de banco de dados não configurado para
   esta tecnologia');
 end;
 Result := FBancoDados;
end;

function TDBXFactory.CriaDataset: IDataset;
begin
 Result := TNossoDbxDataset.Create;
 Result.SetBancoDados(CriaBancoDados as TSQLConnection);
end;

destructor TDBXFactory.Destroy;
begin
 FBancoDados.Free;
 inherited;
end;

end.
```

Listagem 7. Classe TBdFactory

```
unit uBdFactory;

interface

uses
  uAbstractFactory, uIDataset;

type
  TBdFactory = class
    class var FFactory: TAbstractFactory;
  public
    class procedure InicializaBD;
    class procedure FinalizaBD;
    //essas duas functions são meramente atalhos
    class function CriaBancoDados: TObject;
    class function CriaDataset: IDataset;
    class property BdFactory: TAbstractFactory read FFactory;
  end;

implementation

uses
  IniFiles, SysUtils, forms, uADOFactory, uDBXFactory;

{ TBdFactory }

class function TBdFactory.CriaBancoDados: TObject;
begin
  Result := FFactory.CriaBancoDados;
end;

class function TBdFactory.CriaDataset: IDataset;
begin
  Result := FFactory.CriaDataset;
end;

class procedure TBdFactory.FinalizaBD;
begin
  FFactory.Free;
end;

class procedure TBdFactory.InicializaBD;
var Tecnologia: string;
begin
  with TIniFile.Create(ExtractFilePath(Application.ExeName)+'config.ini')
do
  try
    Tecnologia := ReadString('config', 'Tecnologia', '');
    if Tecnologia = 'DBX' then
      FFactory := TDBXFactory.Create
    else
      if Tecnologia = 'ADO' then
        FFactory := TADOFactory.Create
      else
        raise Exception.Create('Tecnologia desconhecida!');
        FFactory.TipoBancoDados := ReadString(Tecnologia, 'TipoBancoDados', '');
    if FFactory.TipoBancoDados = '' then
      raise Exception.Create('Tipo de banco de dados não encontrado ou
      não configurado para essa tecnologia.');
      FFactory.Servidor := ReadString(FFactory.TipoBancoDados, 'Servidor', '');
      FFactory.NomeBanco := ReadString(FFactory.TipoBancoDados,
      'NomeBanco', '');
      FFactory.Login := ReadString(FFactory.TipoBancoDados, 'Login', '');
      FFactory.Senha := ReadString(FFactory.TipoBancoDados, 'Senha', '');
  finally
    Free;
  end;
end;

initialization
  TBdFactory.InicializaBD;

finalization
  TBdFactory.FinalizaBD;

end.
```

Listagem 8. Firebird

```
/*criação da tabela*/
CREATE TABLE CONTATOS (
IDContato int not null,
Nome varchar(80),
Telefone varchar(20),

constraint pk_CONTATOS primary key (IDContato)
);

/*generator da tabela*/
CREATE GENERATOR GNCONTATOS;
COMMIT;

/*trigger de autoincremento na inserção*/
CREATE TRIGGER "INSCONTATOS" FOR "CONTATOS"
ACTIVE BEFORE INSERT POSITION 0
AS
    BEGIN
        NEW.IDContato = GEN_ID(GNCONTATOS, 1);
    END
```

Listagem 9. SQL Server

```
CREATE TABLE dbo.Contatos
    (
        IDContato int NOT NULL IDENTITY (1, 1),
        Nome varchar(80),
        Telefone varchar(20),
         constraint PK_Contatos PRIMARY KEY (IDContato)
    )
```

Listagem 10. Arquivo INI

```
[config]
;Tecnologia=ADO
Tecnologia=DBX

[ADO]
;TipoBancoDados=SQLServer
TipoBancoDados=Firebird

[DBX]
;TipoBancoDados=SQLServer
TipoBancoDados=Firebird

[Firebird]
Servidor=localhost
NomeBanco=D:\Documentos\Vitor\ClubeDelphi\FactoryMethod_
Exemplo_11_MultiBanco\Bancos\MultiBanco.fdb
Login=sysdba
Senha=masterkey

[SQLServer]
Servidor=localhost\SQLExpress
NomeBanco=MultiBanco
Login=sa
Senha=123456
```

Nota do DevMan

A carga de dados (parâmetros) do dbExpress através de arquivos .ini não é uma das novidades e já é implementada pelo próprio DBX através do método LoadParamsFromIniFile("arquivo ini"), ou da propriedade LoadParamsOnConnect. Por padrão o dbExpress tende a procurar as configurações de conexões no arquivo dbxConnections.ini que possui toda a estrutura necessária para que seja possível carregar a conexão definida no ambiente de desenvolvimento. Veja o exemplo a seguir:

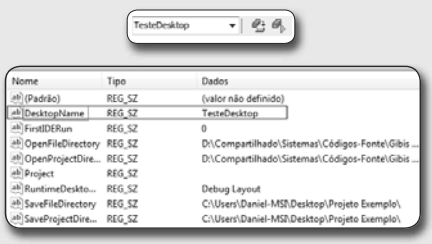
```
try
 SQLConnection1.Params.Clear;
 SQLConnection1.LoadParamsOnConnect := False;
 SQLConnection1.LoadParamsFromIniFile
 ('C:\dbxconnections.ini');
 SQLConnection1.Open;
except
 on e: exception do
 begin
  ShowMessage(e.Message);
 end;
end;
```

Neste caso em que o dbxConnections.ini é considerado como o arquivo de configuração da sua aplicação é bom lembrar que o mesmo deve ser distribuído com seu software. Vale lembrar que é perfeitamente possível criar seu próprio arquivo .ini (usando a unit IniFiles) ou mesmo utilizar as extensões do ClientDataSet como o .xml ou .cds, bastando apenas configurar os parâmetros da sua conexão da seguinte forma:

```
SQLConnection.Params.Values['Nome do Parâmetro'] := Valor;
```

Nota do DevMan

Uma outra maneira de persistir informações referentes a configurações, variáveis e parâmetros é utilizando o próprio registro do Windows. Isto é possível a partir da unit Registry que disponibiliza os métodos de gravação para os tipos de dados mais comuns como String, Boolean, Integer, Float e DateTime por exemplo. Um bom exemplo de ilustração do uso da classe TRegistry é o armazenamento de configurações de componentes de uma determinada tela, no registro do Windows. Por exemplo, caso haja um mesmo relatório em modo retrato e paisagem, mas o usuário utiliza apenas o modo retrato, neste caso haveria um RadioGroup com as duas opções e a selecionada pelo usuário viria sempre como padrão. Isto faz com que se mantenha a configuração para diferentes usuários em um mesmo ambiente, sem que haja necessidade de parametrizar algo no sistema. O próprio IDE do Delphi utiliza o registro para gravar informações referentes a configurações do seu ambiente de desenvolvimento (Desktop) como mostrado:



A seguir um exemplo de como gravar/recuperar informações do registro do Windows utilizando a classe TRegistry que cria uma chave de registro:

```
class function TRegisterUtils.BuscaRegistroWin(
 stRegDir, stRegNome: String): String;
var
 Reg: TRegistry;
begin
 Reg := TRegistry.Create;
 Reg.RootKey := HKEY_CURRENT_USER;
 Reg.OpenKey(stRegDir, True);
 Result := Reg.ReadString(stRegNome);
 Reg.Free;
end;
class procedure TRegisterUtils.GravaRegistroWin
(stRegDir, stRegNome,
 stRegConteudo: String);
var
 Reg: TRegistry;
begin
 Reg := TRegistry.Create;
 try
  Reg.RootKey := HKEY_CURRENT_USER;
  if Reg.OpenKey(stRegDir, True) then
   Reg.WriteString(stRegNome, stRegConteudo);
 finally
  Reg.CloseKey;
  Reg.Free;
 end;
end;
```

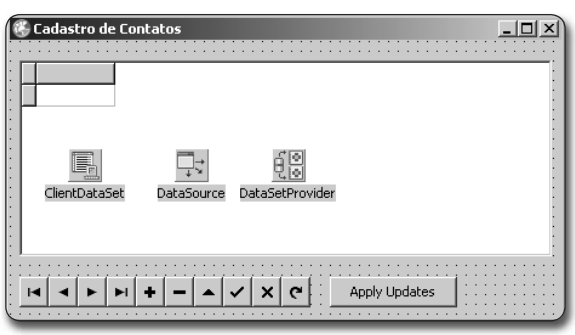


Figura 2. Formulário de cadastro

ção do arquivo INI, como mostram as **Listagens 12** a **15**. As possibilidades podem crescer à medida que são adicionadas fábricas de outras tecnologias, como ZeosDBO ou IBO e à medida que se utilize mais bancos e mais drivers DBExpress e OLEDB.

Como está sendo usado o usuário "SA" para se conectar ao banco de dados, o SQL Server não deve estar configurado apenas como Windows authentication, mas como Windows and SQL authentication. Não é recomendado usar o usuário especial "SA" em aplicações. Esse é apenas um exemplo.

Uma boa pedida para treinar os recursos vistos aqui seria a adição do suporte ao MySQL e drivers OLEDB para outros bancos de dados. Pode-se ainda mesclar o que foi visto aqui com o Framework de persistência via RTTI visto na ClubeDelphi número 113. Também expandir a interface IDataset para que esta possua o nome e valor da chave primária, implementando parâmetros e mestre-detalhes por exemplo.

Pode-se também implementar um componente que encapsule todos os conceitos vistos para poder ter acesso a dados em Design Time (Data Aware). Foi abordado muito neste artigo sobre Singleton e Mediator. Esses são dois outros padrões de projeto muito úteis e necessários.

Conclusão

Espera-se que com esse artigo o leitor esteja apto a criar aplicações multibanco, e "multi-qualquer-coisa" usando Factory Methods e Abstract Factories, afinal esses padrões permitem utilizar várias tecnologias possíveis para um mesmo objetivo, implementando-as ao mesmo tempo, porém, decidindo em tempo de execução qual tecnologia realmente usar.

Também é possível criar um compressor que permita ao usuário utilizar a API do WinZip, 7zip ou outra qualquer, "envelopando" essas APIs com uma interface comum e criando o compressor desejado em tempo de execução via Factory Method.

Outra ideia interessante é criar um game que use tanto OpenGL como DirectX postergando a "Runtime" qual implementação efetivamente usar.

Com base neste artigo também é possível, juntamente com outros conheci-

mentos, criar um Framework de persistência que seja independente de banco de dados, para que qualquer aplicação funcione em qualquer um deles.

Com isso qualquer sistema nunca mais ficará limitado a um único banco de dados. Se for necessário, recomece um sistema do zero usando esse artigo como ponto de partida. O tempo em que se alcançará o número de funcionalidades do projeto sistema atual será menor, e com menos bugs, e o aprendizado permitirá o "know how" necessário para criar sistemas ainda mais flexíveis.

Padrões de projeto são viciantes e fontes de assuntos inesgotáveis. Quando se implementa um, logo quer se implementar todos.

Listagem 11. Form de Cadastro

```
unit uFrmCadastro;

interface

uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
 Dialogs, StdCtrls, DB, DBClient, Provider, ExtCtrls, DBCtrls, Grids, DBGrids,
 uIDataset, uBdFactory;

type
 TfrmCadastro = class(TForm)
   DBGrid1: TDBGrid;
   DBNavigator1: TDBNavigator;
   DataSource1: TDataSource;
   DataSetProvider1: TDataSetProvider;
   ClientDataSet1: TClientDataSet;
   btSalvar: TButton;
   procedure btSalvarClick(Sender: TObject);
   procedure FormCreate(Sender: TObject);
 private
   Fdataset: IDataset;
 public
   { Public declarations }
 end;

implementation

{$R *.dfm}

procedure TfrmCadastro.btSalvarClick(Sender: TObject);
begin
 ClientDataSet1.ApplyUpdates(0);
end;

procedure TfrmCadastro.FormCreate(Sender: TObject);
begin
 Fdataset := TBdFactory.CriaDataset;
 Fdataset.SetSql('select  Nome, Telefone from contatos');
 Fdataset.ConfiguraProvider(DataSetProvider1);
 ClientDataSet1.Open;
end;

end.
```

Listagem 12. Firebird com DBX

```
[config]
Tecnologia=DBX

[DBX]
TipoBancoDados=Firebird

[Firebird]
Servidor=localhost
NomeBanco=D:\Documentos\Vitor\Documentos\My Dropbox\Trabalhos_Andamento\ClubeDelphi\FactoryMethod_Exemplo_11_MultiBanco\Bancos\MultiBanco.fdb
Login=sysdba
Senha=masterkey
```

Listagem 13. Firebird com ADO e ODBC

```
[config]
Tecnologia=ADO

[ADO]
TipoBancoDados=Firebird

[Firebird]
Servidor=localhost
NomeBanco=D:\Documentos\Vitor\Documentos\My Dropbox\Trabalhos_Andamento\ClubeDelphi\FactoryMethod_Exemplo_11_MultiBanco\Bancos\MultiBanco.fdb
Login=sysdba
Senha=masterkey
```

Listagem 14. SQL Server com DBX

```
[config]
Tecnologia=DBX

[DBX]
TipoBancoDados=SQLServer

[SQLServer]
Servidor=localhost\SQLExpress
NomeBanco=MultiBanco
Login=sa
Senha=123456
```

Listagem 15. SQL Server com ADO

```
[config]
Tecnologia=ADO

[ADO]
TipoBancoDados=SQLServer

[SQLServer]
Servidor=localhost\SQLExpress
NomeBanco=MultiBanco
Login=sa
Senha=123456
```

Sugere-se, para um maior aprofundamento no assunto, a leitura do livro "Padrões de Projeto" (Design Patterns no original) dos autores: Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides, a famosa Gang of Four (GOF). Leia primeiro o capítulo sobre Factory Method e depois sobre Abstract Factory.

Vitor Luiz Rubio
vitorrubio@gmail.com - http://www.vitorrubio.blogspot.com
Tecnólogo em Processamento de Dados pela FATEC-SP e MCP. Atua como Analista de Sistemas Sênior na Editora Revista dos Tribunais. Programa com Delphi desde a versão 3 e com .Net desde a versão 1.1.

Blogs do autor
http://www.vitorrubio.blogspot.com/
http://twitter.com/vitorrubio

Site do Firebird
http://www.firebirdsql.org

Driver OLEDB IbProvider
http://www.ibprovider.com/customer/index.php

Site IB Phoenix – Drivers ODBC para firebird
http://www.ibphoenix.com

SQL Server 2008 Express
http://www.microsoft.com/express/database

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Cursos Online

Assinatura .net magazine PLUS
Mais conteúdo .NET por muito menos!

A Revista **.net Magazine** oferece para seus assinantes uma série de Cursos Online de alto padrão de qualidade .

Conheça abaixo os cursos já disponíveis:
www.devmedia.com.br/curso/netmagazine

- **Crie uma loja Virtual completa**
- **Construindo relatórios com Crystal Reports e Visual Studio 2005**
- **Criando uma aplicação Web Completa**
- **Criando uma aplicação client/server no Visual Studio 2005**
- **Aprenda a criar um blog com ASP.NET**
- **Curso de C#**

A sua melhor opção de aprendizagem!
Assine a **.net Magazine** e
Comece já seu treinamento!
www.devmedia.com.br/assine

DevMedia Mais Informações: www.devmedia.com.br/central - Tel.: 21 3382-5038

Delphi Prism e ASP.NET - Parte 7

Guinter Pauli

Nesta última etapa, o usuário poderá informar seus dados adicionais para o envio da compra, para isso será usado o recurso de Profiles e um recurso interessante do CreateUserWizard, que é a possibilidade de incluir novos "steps" ao assistente. Os dados que até agora estão na sessão do usuário, em memória, serão finalmente persistidos no banco de dados criado na primeira parte desta série. O processo é simples, varrer o DataSet em memória e gerar comandos SQL de inserção. Será usado ADO.NET diretamente no form, mas considere o uso de classes separadas para criar acesso a dados em uma situação real, ou mesmo o uso de um framework de persistência. O ADO.NET já foi amplamente abordado em etapas anteriores, de forma que não será discutido aqui. O exemplo emprega classes já discutidas, como SqlConnection, SqlCommand, DataSet e DataRow. Observações serão feitas quando pertinente, sempre comparando com o Delphi Win32 tradicional e VCL, objetivo principal desta série.

Profiles

Uma das técnicas mais interessantes que este exemplo usa é o recurso de Profiles. O ASP.NET possui inúmeros recursos para manter o estado em aplicações Web, como

Resumo DevMan

De que trata o artigo:

Lançado pela Embarcadero, o Delphi Prism é a nova forma de desenvolver aplicações para o .NET Framework e Mono utilizando o Object Pascal e o IDE do Visual Studio / MonoDevelop. Nesta série de artigos, veremos como utilizar o Delphi Prism para criar um site de E-Commerce usando a tecnologia ASP.NET.

Para que serve:

O objetivo principal desta série de artigos é familiarizar os desenvolvedores Delphi Win32 com o ambiente de desenvolvimento do Delphi Prism, o IDE do Visual Studio, bem como introduzir o desenvolvimento de aplicações Web. Dessa forma, faremos vários comparativos entre os dois IDEs e também entre os Frameworks ASP.NET e VCL, servindo como um excelente guia para diminuir a curva de aprendizado ao se aprender ASP.NET baseando-se nos seus prévios conhecimentos em Delphi Win32 (Desktop).

Em que situação o tema é útil:

Esta série será muito útil para quem nunca desenvolveu aplicativo algum para Web, nem mesmo teve um primeiro contato com o Delphi Prism / Visual Studio. Ao desenvolver aplicações para Web, ganhamos inúmeras vantagens se compararmos ao ambiente Desktop: Deploy simplificado, uso de computação nas nuvens, não há necessidade de instalações visto que os usuários acessam um mesmo servidor na internet, não há dependência do sistema operacional, pois usamos um browser, não há preocupação com backups nos clientes, nem altos custos para manutenções e visitas aos clientes etc.

Delphi Prism e ASP. NET:

No primeiro artigo desta série tivemos uma introdução ao Delphi Prism, seus objetivos e propósitos. A seguir partimos para a criação do banco de dados no SQL Server usado ao longo desta série. Na segunda parte iniciamos a criação do Web Site, aproveitando para falar dos fundamentos do ASP.NET e Visual Studio, sempre comparando-os com a VCL e RAD Studio, com o intuito de facilitar o aprendizado. Vimos a estrutura básica de um Web Site e iniciamos a construção do layout padrão do site de E-Commerce (chamado "Computer Store"), bem como o sistema de navegação. Na parte 3, vimos como construir a página principal do site, exibindo a lista de produtos, mostrando como fazer a conexão com o banco de dados. Na parte 4 vimos dois importantes passos do site, tarefas comuns realizadas em qualquer tipo de aplicação Web que trabalhe com banco de dados, como pesquisas com parâmetros. Na parte 5, vimos como adicionar uma das principais funcionalidades do site, a possibilidade do usuário adicionar produtos a um carrinho de compras, onde usamos ADO.NET. Na parte 6, vimos um recurso muito interessante e que possui um excelente suporte pronto e nativo do ASP.NET: segurança com os controles de Login. Nesta 7ª e última parte do mini-curso, a aplicação ASP.NET será finalizada com a construção da página de Checkout (fechamento da compra).

Nota

Como já comentado em partes anteriores, existem excelentes frameworks de persistência e mapeamento objeto / relacional para .NET, como o NHibernate e ADO.NET Entity Framework. Como o objetivo deste mini-curso é justamente reaproveitar conhecimentos do Delphi Win32 para o desenvolvimento Web, serão utilizados recursos que mais se assemelham ao DBExpress e Firebird. Pessoalmente prefiro, em meus projetos, utilizar uma abordagem interessante: separar a aplicação em camadas, de forma que as classes de negócio, apresentação (forms) e domínios (as classes que mapeiam as tabelas do banco) não dependam de uma tecnologia de persistência em específico, usando aí também um pouco de Interfaces, Generics e classes POCO (Plain Old CLR Objects). Dessa forma, é possível desenhar uma solução e aplicar um modelo de persistência de acordo com a necessidade, ou mesmo mudar de tecnologia de acesso sem sofrer um impacto drástico.

Nota

A partir desta etapa, o exemplo demonstrado utiliza o Delphi Prism XE.

o simples uso de Cookies, passando por variáveis de Sessão, Cache, ViewState e mais. Imagine a seguinte situação: o usuário entra no seu site e customiza algumas partes da página, define algumas preferências de navegação, usabilidade etc. Quando ele retornar ao seu site, não importa quanto tempo isso leve, devemos ter uma forma de "lembrar" essas preferências definidas pelo usuário. Antigamente (ASP.NET 1.x), era preciso criar uma tabela no banco de dados com campos que definiam o perfil (profile) do usuário. Após a sua visita, era preciso gravar suas preferências no banco e o ID em um cookie local no browser. Quando ele retornava, informações eram recuperadas a partir do cookie.

Um grande recurso introduzido lá no ASP.NET 2.0 foi o suporte a Profiles. Este recurso faz basicamente o que foi exposto no parágrafo anterior, porém, reduzindo drasticamente o trabalho necessário para realizar esta tarefa.

Com o Delphi Prism XE, pode-se utilizar o recurso de Profiles em aplicações Web com ASP.NET 4.0 e Visual Studio 2010. Basta adicionar os campos desejados no Web.Config, como:

```
<add name="VARIAVEL"/>
```

O Delphi Prism faz um parse no XML, verifica o que foi colocado na sessão Profile (campos) e cria automaticamente uma classe "wrapper" que é injetada na página através da propriedade Profile. Esta é uma classe fortemente tipada. No exemplo dado, é possível fazer o seguinte:

```
Profile.VARIAVEL := TextBox1.Text;
```

Isso difere de Sessions, onde não há tipo específico, sempre temos um Object e não há verificação em tempo de compilação:

```
Session["VARIAVEL"] := TextBox1.Text;
```

No momento que o site é acessado pela primeira vez, o ASP.NET aciona o recurso de Profiles. Ele cria "on-the-fly" um banco de dados prontinho com uma estrutura previamente definida para guardar os dados do profile, chamado ASPNETDB, o mesmo usado para guardar dados de login de usuários, visto no artigo anterior. De praxe, o ASP.NET já grava um cookie no browser para mapear o usuário para o seu respectivo profile no servidor quando ele entrar novamente.

Algumas considerações sobre o recurso:

- Por padrão o recurso usa um banco pré-definido no SQL Server chamado ASPNETDB, mas pode-se criar a estrutura em um banco SQL Server existente;
- Com um pouco mais de trabalho é possível usar outros bancos de dados;
- É possível gravar vários tipos de informações no banco que não só strings, mas coleções completas (por exemplo, um carrinho de compras do usuário) bem como tipos definidos pelo próprio desenvolvedor.

Solicitando informações adicionais do usuário

Mas como usar Profiles na prática no exemplo? No site de e-commerce o usuário precisa informar seus dados para o envio do pedido. Normalmente o usuário tem um endereço padrão cadastrado no site, mas tem a opção de trocá-lo ao finalizar a compra. Por exemplo, ele pode comprar um produto para presente que deve ser entregue em outro endereço. Por esse motivo, na tela de Checkout que será mostrada adiante, é preciso exibir os dados de postagem atualmente cadastrados.

Como visto na parte anterior do artigo, os dados de login do usuário ficam gravados nas "tabelas de sistema" do ASP.NET. É o banco ASPNETDB, que tem uma estrutura pré-definida e juntamente com os controles de login e providers já faz todo o gerenciamento de contas. No exemplo do artigo, como fazer para adicionar mais informações a esse cadastro do usuário, como os dados postais de entrega (endereço, cidade, CEP)? Apesar de possível, o trabalho de modificar diretamente a estrutura da tabela criada no ASPNETDB não é recomendado. Talvez criar uma tabela auxiliar que contenha um join para a tabela de usuário, linkada pelo UserID.

Sem dúvida aqui está um belo exemplo para o uso de Profiles. É justamente isso que o exemplo faz, utiliza o recurso para armazenar os dados adicionais do usuário, além dos dados padrão já requeridos pelo sistema de membership do ASP.NET. Nada melhor do que solicitar essas informações durante o próprio processo de cadastro, na tela NewUser.aspx, no controle Create UserWizard, que agora contém alguns TextBoxs adicionais (**Figura 1**).

Para salvar essas informações no Profile do usuário, basta manipular o evento ContinueButtonClick do CreateUserWizard, como o código da **Listagem 1.** Simplesmente é atribuída a cada propriedade do Profile o texto digitado em cada TextBox. Não é necessário inserir qualquer código para acesso a banco de dados, o ASP.NET faz tudo. Estes dados são salvos no próprio banco ASPNETDB.

No Web.Config os campos utilizados no Profile devem ser informados, para que o ASP.NET possa gerar os dados corretamente no ASPNETDB. A **Listagem 2** exibe as chaves incluídas no arquivo de configuração do Web Site. Não é necessário informar o tipo de dado, pois se ele for string, o ASP.NET é esperto o

suficiente para saber que a maioria dos dados utilizados são deste tipo.

Página de Checkout

A **Figura 2** mostra a página de Checkout do sistema de E-Commerce. É nessa página que o usuário irá confirmar os dados de envio, bem como confirmar o método de pagamento. O exemplo simplesmente simula um pagamento como cartão de crédito, solicitando os dados básicos, já que a implementação completa de um processo de pagamento on-line está fora do escopo deste artigo. Os Labels vistos na tela simplesmente são alimentados com os dados de endereço do usuário, através do Profile. Isso é feito no Page_Load quando a página é carregada, como mostra a **Listagem 3**.

A **Listagem 4** mostra o código do botão Continue. Nele existe a chamada para dois métodos importantes, o SendToDataBase e o SendEmail, apresentados adiante. A seguir, o usuário é redirecionado para a página de finalização que simplesmente exibe uma mensagem de confirmação. Neste ponto, os dados da compra são persistidos da memória para o banco de dados e um email de confirmação com os dados da compra é enviado ao usuário. Observe o uso de Type Inference usando a palavra-reservada var.

Persistindo os dados de sessão no banco de dados

A **Listagem 5** detalha o método SendToDatabase da página de Checkout. O código basicamente usa ADO.NET para criar uma conexão com o banco de dados, para então inserir as informações do pedido, que até agora estão em sessão (memória). A ConnectionString é lida a partir do Web.Config, isso é feito através do ConfigurationManager (ou WebConfigurationManager). Um SqlCommand excuta um comando no servidor para inserir as informações do pedido. Uma Stored Procedure é usada (**Listagem 6**).

O for each percorre o DataSet, que nada mais é que o carrinho de compras em memória e para cada item encontrado executa um insert no banco. A Stored Procedure retorna o ID do pedido, que é usado para ser incluído como valor do campo da chave estrangeira na tabela de detalhes do pedido, fazendo assim o relacionamento correto entre tabelas master e detail.

Enviando um email de confirmação

A **Listagem 7** mostra o código que envia o email ao usuário. O procedimento é semelhante ao mostrado no SendToDataBase, pois é preciso varrer o carrinho para verificar quais itens foram comprados. No entanto, ao invés dos dados serem enviados ao banco, agora o sistema monta um e-mail que utiliza configurações pré-definidas de um servidor SMTP para o envio. Note que o email do usuário é obtido a partir do seu Profile. O StringBuilder é usado para montar o corpo do email.

Teste final

O teste final no exemplo comprova a funcionalidade dos códigos expostos. A **Figura 3** mostra alguns produtos adicionados ao carrinho. Ao fazer o Checkout, a **Figura 4** solicita a confirmação dos dados de envio e o método e pagamento. Finalizando a compra, o usuário recebe um email, como mostra a **Figura 5.**

Nota do DevMan

O controle CreateUserWizard descende de Wizard que foi incluído no ASP.NET 2.0 / Visual Studio 2005, que permite criar um mecanismo "passo a passo" para preenchimento de informações. Inclusive na Web pode ser usado para simular um PageControl. É possível adicionar mais "steps" ao controle e dentro de cada aba adicionar mais controles

Listagem 1. Dados adicionais colocados no CreateUserWizard são salvos no Profile

```
method NewUser.CreateUserWizard1_
ContinueButtonClick(
  sender: System.Object; e: System.
   EventArgs);
begin
  Profile.FullName := TextBox1.Text;
  Profile.Address := TextBox2.Text;
  Profile.City := TextBox3.Text;
  Profile.Country := TextBox4.Text;
  Profile.ZIPCode := TextBox5.Text;
  Profile.EMAIL := CreateUserWizard1.Email;
end;
```

Listagem 2. Profile no Web.Config

```
<profile>
  <properties>
    <add name="FullName"/>
    <add name="Address"/>
    <add name="City"/>
    <add name="Country"/>
    <add name="ZIPCode"/>
    <add name="EMAIL"/>
  </properties>
</profile>
```

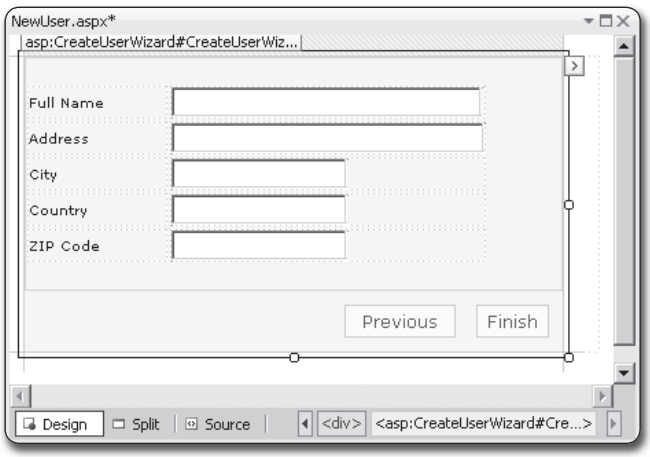


Figura 1. CreateUserWizard com "step" adicional

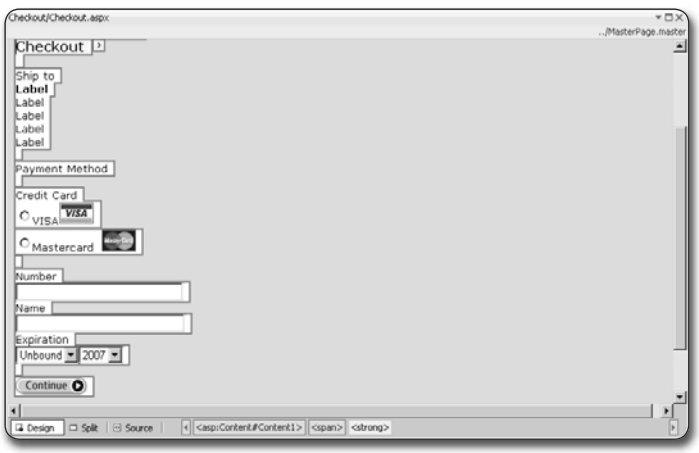


Figura 2. Design da página Checkout.aspx

Listagem 3. Dados do Profile com informações postais são repassados aos Labels no carregamento da página (Page_Load)

```
method Checkout_Checkout.Page_Load(
  sender: Object; e: EventArgs);
begin
  if not IsPostBack then
  begin
    Label1.Text := Profile.FullName;
    Label2.Text := Profile.Address;
    Label3.Text := Profile.City;
    Label4.Text := Profile.Country;
    Label5.Text := Profile.ZIPCode;
    // preenche o combo com 12 meses do ano
    for i: Integer := 1 to 12 do
      DropDownList1.Items.Add
      (Convert.ToString(i.ToString()));
  end;
end;
```

Listagem 4. Finalizando a compra: gravação dos dados no banco e envio do email de confirmação

```
method Checkout_Checkout.Image
Button1_Click(
  sender: System.Object; e: System.Web.
  UI.ImageClickEventArgs);
begin
  var OrderId := SendToDatabase();
  SendEmail(OrderId);
  Response.Redirect("~/Checkout/finish.aspx");
end;
```

Listagem 5. Envio dos dados ao BD a partir das informações em memória

```
method Checkout_Checkout.SendToDatabase(): Integer;
begin
  var OrderId:Object;
  var con := new SqlConnection(
  ConfigurationManager.ConnectionStrings['LOJA'].ConnectionString);
  var cmd := new SqlCommand('InsertOrder', con);
  cmd.CommandType := CommandType.StoredProcedure;
  cmd.Parameters.AddWithValue('@DATE', DateTime.Now);
  cmd.Parameters.AddWithValue('@CUSTOMER', User.Identity.Name);
  cmd.Parameters.Add('@RETURN_VALUE', SqlDbType.Int);
  cmd.Parameters['@RETURN_VALUE'].Direction := ParameterDirection.ReturnValue;
  con.Open();
  try
    cmd.ExecuteNonQuery();
    OrderId := cmd.Parameters['@RETURN_VALUE'].Value;
    var ds := DataSet(Session['ShoppingCart']);
    for each row: DataRow in ds.Tables[0].Rows do
    begin
      var SQL := System.String.Format(
        'insert into OrderDetails (OrderId,ProductId) values ({0},{1});', OrderId, row['ProductId']);
      var cmd2 := new SqlCommand(SQL, con);
      cmd2.ExecuteNonQuery()
    end
  finally
    con.Close()
  end;
  result := Convert.ToInt32(OrderId);
end;
```

Nota do DevMan

Existe um novo recurso no Visual Studio 2010 que pessoalmente me agradou muito. Na verdade ele teve outro propósito, mas "por tabela" acabou agradando também a muitos outros desenvolvedores que estavam acostumados com o IDE do Delphi onde as janelas e forms (no estilo Delphi 7) ficam soltas em tela, "Classic Unlocked". O Visual Studio 2010 permite que o desenvolvedor arraste uma janela, a partir de sua barra superior, e a retire (literalmente "desgrude") do editor principal, deixando-a totalmente solta em tela para ser colocada onde quiser, exatamente os bons e velhos TForms do Delphi 7. A equipe da MS fez isso no Visual Studio 2010 para permitir que o desenvolvedor arraste o designer (ou editor, ou outra janela qualquer) para um segundo monitor, podendo programar em ambos, para Windows que suportam monitores estendidos. Uma ótima ideia que teve um duplo benefício. Além disso, você fica com muito mais espaço para organizar seus controles no form.

Nota do DevMan

O "for each" do Delphi Prism é análogo ao "for each" do C# e ao "for in" do Delphi Win32. Permite varrer uma coleção sem que para isso sejam conhecidos os seus limites, muito útil para evitar acesso que extrapole os limites do array / coleção. O Prism inclusive tem mais algumas variações para o "for each", como "for each matching" para considerar objetos somente de um tipo. E uma curiosidade, é possível omitir o "each" e obter o mesmo resultado, algo como "for objeto in coleção".

Nota do DevMan

Type Inference

No Delphi Win32 é preciso sempre declarar o tipo de uma variável. Quando for um objeto, deve ser instanciado com Create chamando o construtor da classe. O código a seguir declara um TClientDataSet e a seguir o instancia. Mas veja bem, por que é preciso declarar a classe como sendo TClientDataSet e depois no momento da criação atribuir um objeto do mesmo tipo à variável?

```
var
 Cds: TClientDataSet;
begin
 Cds := TClientDataSet.Create(nil);
```

Isso tem uma explicação. Em alguns casos, uma variável não é declarada do mesmo tipo que vai ser instanciada, mas sim de um tipo ancestral. Assim, é possível aplicar técnicas de orientação a objeto como polimorfismo e abstração, além de programação para interfaces. É por esse motivo, por exemplo, que o Quick Report aponta para um TDataSet e até hoje pode suportar inúmeros engines de acesso a dados, inclusive os que venham a ser criados.

O código a seguir declara uma variável do tipo TDataSet, classe base para muitas outras classes de acesso a dados na VCL. Porém, na implementação essa variável pode receber tanto um TQuery quanto um TClientDataSet, pois ambos descendem de TDataSet. Quando o método Post do DataSet é chamado, o compilador vai analisar a assinatura do método na classe declarada (TDataSet), e vai detectar que ele é virtual. Com isso, o verdadeiro Post que será chamado não é o da classe TDataSet, mas de TClientDataSet, que sobrescreve o Post com override, configurando uma chamada polimórfica:

```
var
  Ds: TDataSet;
begin
  if [condicao] then
    Ds := TQuery.Create(nil)
  else
    Ds := TClientDataSet.Create(nil);
  Ds.Post();
```

E no Delphi Prism? Bem, é possível fazer tudo isso sem problemas no Prism, usando objetos da FCL ao invés da VCL, claro. Porém, aqui existe uma novidade. Se não formos usar polimorfismo, ou seja, o tipo que vamos instanciar é o mesmo que vai ser declarado (Inline ou na sessão var), pode-se omitir o tipo:

```
// Declarando o tipo
var Ds1: DataSet := new DataSet();
// Type Inference, sem declarar o tipo
var Ds2 := new DataSet();
// Outros exemplos de Type Inference
var i := 32;
var s := "Guinther";
```

No primeiro código, um trecho normal, usando apenas variáveis Inline (o Prism permite declarar variáveis em qualquer lugar do código, como no C#, não precisa sessão var). O objeto é declarado como DataSet e instanciado a partir do construtor da classe (a palavra-chave para instanciar objetos é NEW, e não Create). No segundo código o tipo foi omitido, ou seja, foi declarado sem tipo. O compilador do Delphi Prism, a partir do valor que for jogado dentro da variável Ds2, determina o tipo do objeto. Na sequência, outros exemplos de como usar o recurso. Isso é chamado de Type Inference (não são Variants não, são variáveis fortemente tipadas).

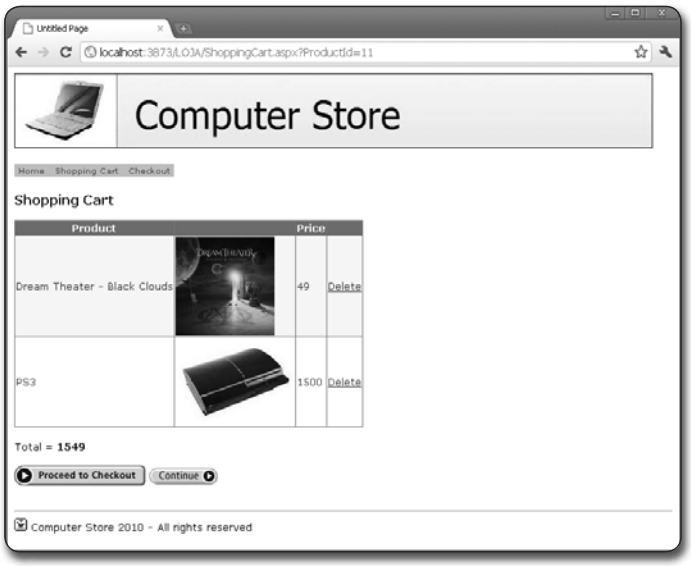


Figura 3. Itens da compra (carrinho) ficam em memória, no DataSet

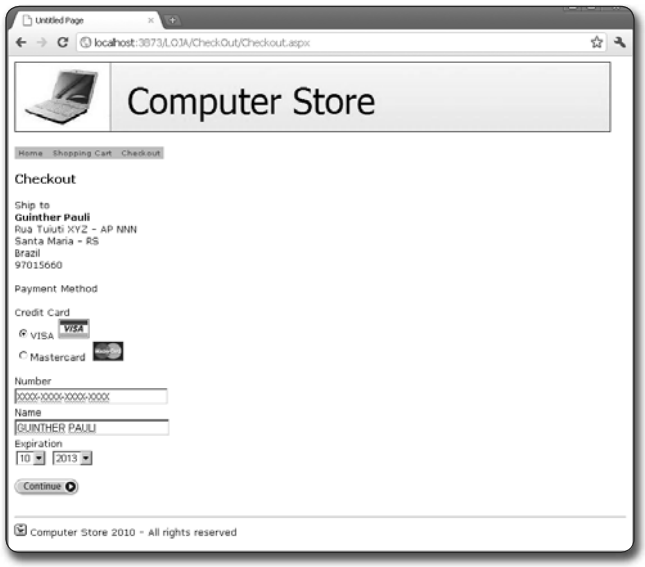


Figura 4. Confirmação dos dados para envio e pagamento

Listagem 6. Stored Procedure para inserir pedidos no banco

```
create procedure InsertOrder
 @date datetime,
 @customer varchar(50)
as
begin
  insert into Orders (date,customer)
  values (@date,@customer);
  return @@IDENTITY;
end
```

Listagem 7. Enviando ao usuário um email de confirmação da compra

```
method Checkout_Checkout.SendEmail(OrderId: Integer);
begin
  var msg := new System.Web.Mail.MailMessage();
  msg.From := 'guinther@devmedia.com.br';
  msg.To := Profile.EMAIL;
  msg.Subject := 'Your order';
  var sb := new StringBuilder();
  sb.AppendLine('Order Number: ' + OrderId.ToString());
  sb.AppendLine('');
  var ds: DataSet := Session['ShoppingCart'] as DataSet;
  for each row: DataRow in ds.Tables[0].Rows do
  begin
    sb.AppendLine('Product: ' + row['ProductName'].ToString());
    sb.AppendLine('Price: ' + row['Price'].ToString());
    sb.AppendLine('')
  end;
  sb.AppendLine('Total: ' + ds.Tables[0].Rows[0]['Total'].ToString());
  msg.Body := sb.ToString();
  msg.BodyFormat := MailFormat.Text;
  var UserName := 'coloque aqui o email do usuário SMTP – ex.: usuario@empresa.com';
  var Password := 'coloque aqui a senha do usuário SMTP';
  var MailServer := 'coloque aqui o endereço smtp do servidor – ex.: smtp.empresa.com;
  msg.Fields.Add('http://schemas.microsoft.com/cdo/configuration/smtpauthenticate', 1);
  msg.Fields.Add('http://schemas.microsoft.com/cdo/configuration/sendusername', UserName);
  msg.Fields.Add('http://schemas.microsoft.com/cdo/configuration/sendpassword', Password);
  msg.Fields.Add('http://schemas.microsoft.com/cdo/configuration/sendusing', 2);
  msg.Fields.Add('http://schemas.microsoft.com/cdo/configuration/smtpserver', MailServer);
  msg.Fields.Add('http://schemas.microsoft.com/cdo/configuration/smtpconnectiontimeout', 10);
  msg.Fields.Add('http://schemas.microsoft.com/cdo/configuration/smtpserverport', 587);
  msg.Fields.Add('http://schemas.microsoft.com/cdo/configuration/smtpusessl', false);
  System.Web.Mail.SmtpMail.Send(msg)
end;
```

Nota do DevMan

Muitos desenvolvedores Delphi têm o costume de utilizar o Page_Load como se fosse o OnShow de um TForm, sendo essa uma das principais fontes para bugs em sistemas Web. É importante levar em consideração que o Page_Load é executado a cada vez que há uma carga completa da página, no caso do ASP.NET tradicional (sem AJAX). Isso ocorre toda vez que um usuário clica em um botão, em uma célula de um GridView que contenha botões, LinkButtons ou em controles que estejam com o AutoPostBack ativado. Dessa forma, para simular realmente um OnShow, garantindo que o código será executado apenas uma vez, quando a página é carregada, é preciso verificar o valor da propriedade IsPostBack. Ele será false no primeiro carregamento.

Nota do DevMan

Um StringBuilder do .NET Framework é semelhante a um TStringList da VCL. É usado para concatenar strings. No .NET, quando strings são concatenadas com o operador "+", a plataforma simplesmente descarta a string original e cria outra em memória, ao contrário do Delphi Win32, onde uma string cresce dinamicamente. Isso causa sérios problemas de performance, e na Web cada milissegundo perdido pode se tornar uma grande dor de cabeça em um Web Site com centenas ou milhares de conexões simultâneas. O StringBuilder resolve o problema e é tão simples de usar quanto o TStringList a VCL, senão análogo.

Nota do DevMan

O SQL Server permite que o desenvolvedor obtenha o ID gerado pelo banco de dados para uma tabela que tenha um campo Identity (autoincremento), através da variável de contexto @@IDENTITY. É semelhante a um recurso utilizado no Firebird para descobrir o valor gerado por uma Sequence / Generator. Para obter esse valor no código, basta recuperar o valor de @RETURN_VALUE, obtido como parâmetro de saída (Output) de um SqlCommand.

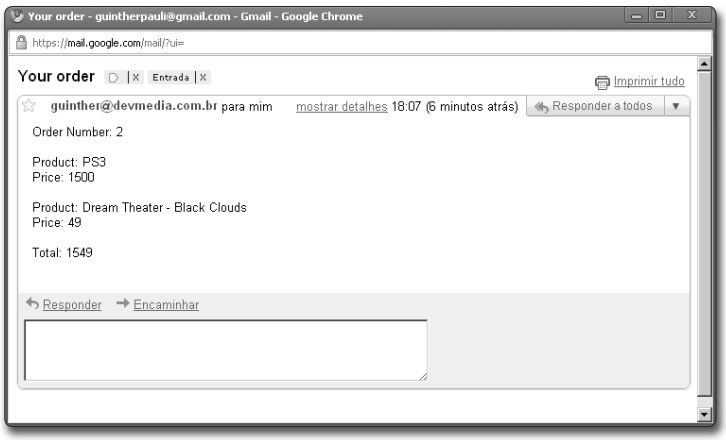


Figura 5. Usuário recebe e-mail com confirmação da compra e descrição dos itens

Nota

Este mini-curso não encerra de forma alguma nossa cobertura sobre os recursos do Delphi Prism e .NET, muita coisa vem pela frente: vamos abordar em futuros artigos algumas novas tecnologias como WCF, WPF, Silverlight, ASP.NET AJAX 4, ADO.NET Entity Framework, ASP.NET MVC, Windows Azure e muito mais.

Conclusão

Ao longo desta série muitos recursos do Visual Studio, Delphi Prism e ASP.NET foram explorados, sempre confrontando com as metodologias tradicionais do Delphi Win32. Não tenho dúvida que o leitor constatou que desenvolver para Web, com ASP.NET e Delphi Prism, é tão simples quanto desenvolver aplicações Desktop, podendo ainda aproveitar muitos conhecimentos já adquiridos. Desejo aos novos desenvolvedores Web sucesso em seus novos projetos com Delphi Prism e ASP.NET.

Guinther Pauli
guinther@devmedia.com.br
guinther.pauli@gmail.com
http://twitter.com/guintherpauli
http://gpauli.com

Desenvolve em Delphi há 15 anos, tendo utilizado a ferramenta desde a versão 1, passando pelo Turbo Pascal e Borland Pascal for Windows. Certificado em Delphi 3, 5, 6, 7, 2005, 2006, Delphi para Web e Delphi para Linux, palestrante em 5 edições da Borland Conference. Certificado Microsoft MCP, MCAD, MCSD.NET, MCTS e MCPD (C#, ASP.NET), utilizando a plataforma .NET desde a versão beta. Editor das Revistas ClubeDelphi, .NET Magazine Brasil e EASY .NET Magazine, professor de Pós-Graduação da UNIVEM-SP, na área .NET. Especialista em desenvolvimento Delphi, C#, .NET, ASP.NET, Visual Studio, atuando também em projetos de migração, orientando empresas e desenvolvedores na adoção de melhores práticas, ministra treinamentos em .NET focados em equipes Delphi. Foi arquiteto e desenvolvedor ASP.NET do site de doações on-line do projeto Criança Esperança da Rede Globo / Unesco.

Site oficial do Delphi Prism
http://www.embarcadero.com/products/delphi-prism

Wiki do Delphi Prism
http://prismwiki.codegear.com/en/Main_Page

Download do Delphi Prism
https://downloads.embarcadero.com/free/delphi_prism

Site oficial do ASP.NET
http://www.asp.net

Visual Studio
www.microsoft.com/visualstudio

Blogs do autor
http://twitter.com/guintherpauli
http://gpauli.com

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Prol EDITORA GRÁFICA

*Sr. Cliente favor ler as informações com atenção, dúvidas entrar em contato com atendimento.

Relatório de Pré-Flight

OP: 88117 Cliente: Neoficio Data: 25/11/10 Operador: Ayres

Informações da OP:
Título: Clube Delphi 124
Quantidade de Páginas: 48 + Capa
Formato do Miolo: 205x275 mm
Formato da Capa: 205x275 mm
Tamanho da Lombada:-
Tipo de Acabamento: Canoa

Informações do Arquivo:
Título: Clube Delphi 124
Quantidade de Páginas: 48 + Capa
Formato do Miolo: 205x275 mm
Formato da Capa: 205x275 mm
Tamanho da Lombada:-
Tipo de Acabamento: Canoa

Ocorrências do Arquivo

•Imagens com resolução abaixo de **200dpi**:

•Imagens com resolução abaixo de **100dpi**:

03 a 22/24 a 34/42/44/45/47/49/50/Capa 02

•Texto Preto nas 4 Cores:

Capa 02

•Imagens em RGB nas páginas:

Capa 03

•Páginas sem sangria e/ou inferior a 3 mm: (Filete branco no acabamento)

•Texto rente ao corte Inferior a 5 mm e/ou cortando:

08/24

•Colofon: **OK**

• Código de Barra: **OK**

Observações:

OBS: As informações em **VERMELHO** necessitam de atenção, ***se de acordo assinar.***

Data ____/____/____ Cliente: _____

Nome Legível